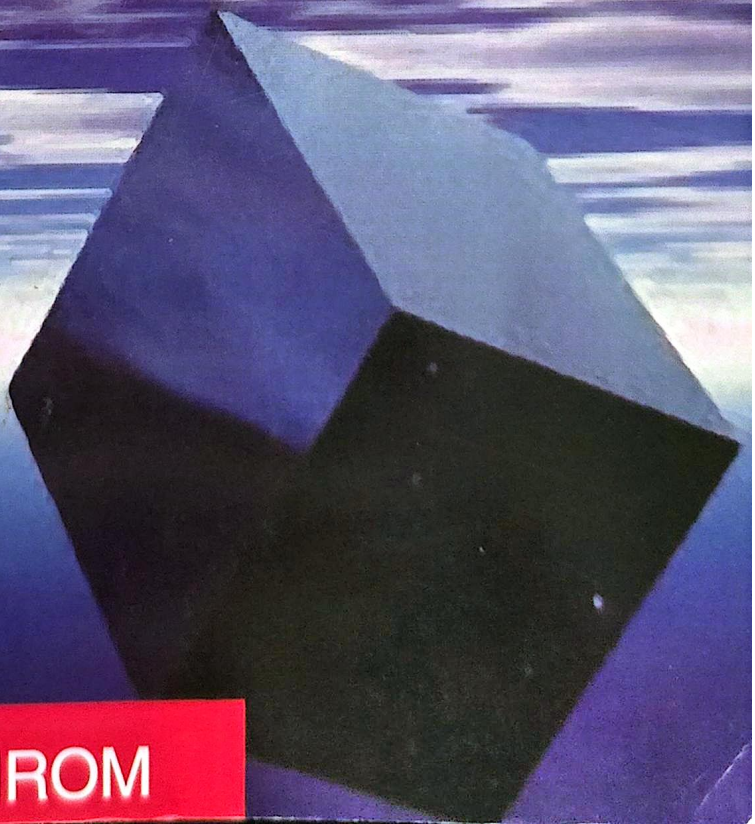
VOLUM COORDONAT DE

SABIN BURAGA

Situri WEB la cheie

SOLUȚII PROFESIONALE DE IMPLEMENTARE

- PHP
- Perl
- ASP.NET
- ActionScript
- XML
- SQL
- Palm OS



POLIROM

SABIN BURAGA
(coord.)

Situri Web la cheie

Seria *Web* este coordonată de dr. Sabin Buraga
(Facultatea de Informatică, Universitatea „Al.I. Cuza", Iaşi)

Sabin Buraga, lector doctor la Facultatea de Informatică a Universităţii „Al.I. Cuza" din Iaşi, este titularul cursurilor *Tehnologii Web*, *Reţele de calculatoare*, *Semantic Web* şi *Interacţiune om-calculator*. Coordonează un grup de interes în domeniul tehnologiilor Web şi are ca direcţii de cercetare Web-ul semantic şi interacţiunile între sisteme distribuite pe scară largă. Mai multe detalii privind activităţile pe care le desfăşoară sunt disponibile pe Web la **http://www.infoiasi.ro/~busaco/**. De acelaşi autor, la Editura Polirom au apărut: *Utilizare Linux* (colaborare cu Dragoş Acostăchioaie, 2004), *Aplicaţii Web la cheie* (coord., 2003), *Proiectarea siturilor Web* (2002), *Programare Web în bash şi Perl* (colaborare cu V. Tarhon-Onu şi Şt. Tanasă, 2002), *Atelier de programare în reţele de calculatoare* (colaborare cu G. Ciobanu, 2001).

© 2004 by Editura POLIROM

www.polirom.ro

Editura POLIROM
Iaşi, B-dul Carol I nr. 4, P.O. BOX 266, 700506
Bucureşti, B-dul I.C. Brătianu nr. 6, et. 7, ap. 33, O.P. 37; P.O. BOX 1-728, 030174

Descrierea CIP a Bibliotecii Naţionale a României:

Situri Web la cheie: soluţii profesionale de implementare/ Sabin Buraga (coord.), Viorel Atofani, Sorin Bahmătă, Marian Bostan, ... – Iaşi: Polirom, 2004

368 p.; 24 cm

ISBN: 973-681-785-7

I. Buraga, Sabin Corneliu (coord.)
II. Atofani, Viorel
III. Bahmătă, Sorin
IV. Bostan, Marian

004.738.52

Printed in ROMANIA

Sabin Buraga
(coord.)

Viorel Atofani, Sorin Bahmătă, Marian Bostan,
Cătălin Bulancea, Andrei Croitoru, Adrian Lazariuc,
Bogdan Manolache, Daniel Puiu, Cosmin Vârlan

Situri Web la cheie

Soluții profesionale de implementare

POLIROM
2004

Cuprins

Prefață (Sabin Buraga) 7

I. România în Flash (Andrei Croitoru) 11

1. Introducere 11
2. Prezentare generală a mediului Flash MX 12
3. Descrierea aplicației 20
4. Detalii de implementare 22
5. Comentarii bibliografice 42

II. De la Flash la PHP. Dezvoltarea unui forum Web (Cosmin Vârlan) 45

1. Descrierea forumului 45
2. Realizarea în PHP a aplicației care rulează pe server 46
3. Realizarea în Flash a aplicației care rulează la client 55
4. Concluzii 75
5. Comentarii bibliografice 76

III. Portaluri educaționale (Bogdan Manolache) 77

1. La început de capitol 77
2. Portaluri 78
3. Studiu de caz: biblioteca virtuală 85
4. Concluzii 100
5. Comentarii bibliografice 101

IV. Modelarea și reprezentarea relațiilor dintre tabele (Sorin Bahmătă) 103

1. Preambul 103
2. Relațiile dintre tabele 103
3. Descrierea metadatelor 112
4. Descrierea aplicației *DataR* 116
5. Manual de utilizare 131
6. Comentarii bibliografice 134

V. Noua provocare în grafica 2D – SVG (Cătălin Bulancea) 135

1. Introducere 135
2. Mai multe despre SVG 136
3. Descrierea aplicației *RelVis* 151
4. Comentarii bibliografice 178

VI. *FindMyHome*, o soluție *e-business* în domeniul imobiliar (Viorel Atofani) 181
1. Introducere 181
2. Premise și obiective 182
3. Proiectarea aplicației 184
4. Interfața pentru clienți 189
5. Maniera de administrare a aplicației 195
6. Concluzii 209
7. Comentarii bibliografice 209

VII. Managementul *blog*-urilor în Perl (Sabin Buraga) 211
1. Preliminarii 211
2. Ce este un *Weblog*? 212
3. Implementarea aplicației *CSBlog* 214
4. Alternative și posibile extinderi viitoare 233
5. Comentarii bibliografice 233

VIII. *Hattrick* – simularea meciurilor de fotbal pe Web (Daniel Puiu, Marian Bostan) 235
1. Fotbalul, între realitate și simulare 235
2. Simularea meciului – prezentare generală 236
3. *Hattrick* – manager de fotbal pe Web 264
4. Concluzii 297
5. Comentarii bibliografice 298
Anexă: Structura tabelelor folosite de aplicația *Hattrick* 299

IX. De la *desktop* la dispozitive *wireless* (Adrian Lazariuc) 303
1. Tehnologia *handheld* 303
2. Proiectarea aplicațiilor Palm OS 310
3. Sincronizarea cu calculatorul: proiectarea conduitelor 328
4. Aplicația practică *PalmNotes* 331
5. Comentarii bibliografice 345

Anexa I. ShockWave Flash (.SWF) *versus* Scalable Vector Graphics (.SVG) (Andrei Croitoru, Cătălin Bulancea) 347
Exemple 353

Anexa II. PHP *versus* ASP.NET (Bogdan Manolache, Sabin Buraga) 357
Ce este PHP? 357
Ce este ASP.NET? 359
Ce alegem? 360

Bibliografie generală 363

Prefaţă

Orice tehnologie suficient de avansată nu se distinge de magie.
Arthur C. Clarke

Spaţiul World Wide Web a devenit de o diversitate şi de o dinamică aproape ameţitoare şi, totodată, magică. Totuşi, analizându-i evoluţia, constatăm faptul că – mai ales în ceea ce priveşte maniera de proiectare şi dezvoltare a siturilor – apar o serie de modele (şabloane, reţete, trucuri) şi un număr relativ restrâns de limbaje şi tehnologii folosite pe scară largă. Pe unele dintre acestea, prin parcurgerea volumului de faţă, cititorul va avea ocazia să le cunoască, urmând ca ulterior să le aprofundeze şi/sau să le utilizeze.

Astfel, putem considera conţinutul acestei cărţi – făcând parte din seria *Web* a Editurii Polirom – ca fiind o radiografie a tehnologiilor actuale de programare Web, descriind metodologiile şi cele mai bune practici în clădirea unor aplicaţii Web concrete, de anvergură. Materialul continuă tematica abordată de volumul anterior – S. Buraga (coord.), *Aplicaţii Web la cheie*, Polirom, 2003 – printr-o suită mai largă de soluţii, recurgând la limbaje şi paradigme de programare Web precum PHP, ActionScript, C#, Perl sau JavaScript.

Cele nouă aplicaţii originale – avându-i drept autori, cu o singură excepţie, pe absolvenţii Facultăţii de Informatică a Universităţii „Al.I. Cuza" din Iaşi – detaliază etapele de proiectare, implementare şi exploatare a unor tipuri reprezentative de situri Web, într-o varietate de domenii (multimedia, *e-business*, comunităţi virtuale, *e-learning*, Web semantic, *wireless* etc.).

Arhitectura volumului

Ne exprimăm speranţa ca informaţiile puse la dispoziţie de capitolele descrise succint în continuare să suscite interesul actualilor sau viitorilor „magicieni" Web.

Primul capitol se concentrează asupra prezentării manierei de realizare în Flash a unei enciclopedii multimedia privitoare la România. *Andrei Croitoru*, după o scurtă descriere a mediului Flash MX şi a limbajului ActionScript, demonstrează că această tehnologie, mai ales prin posibilităţile de *scripting* oferite, poate fi folosită cu succes la implementarea unei aplicaţii interactive

modulare și flexibile, exploatabile grație meta-limbajului *XML* (*Extensible Markup Language*). Una dintre principalele atracții ale capitolului este, probabil, generarea dinamică a drumurilor de pe teritoriul României, prin intermediul algoritmului de determinare a drumului de cost minim dintr-un graf neorientat.

Nu ne despărțim foarte repede de Flash, pentru că al doilea capitol – redactat de *Cosmin Vârlan* – furnizează o soluție de implementare a unui forum prin integrarea a două tehnologii aparent incompatibile: Flash MX 2004 (ActionScript 2.0), pe partea client, și PHP 4 la nivel de server. Datele transmise între client și server sunt structurate ca fișiere XML, pentru stocarea efectivă pe server apelându-se la baze de date relaționale MySQL. Programatorii vor afla în detaliu pașii care trebuie urmați pentru crearea unui forum complet funcțional.

În cel de-al treilea capitol, ne (re)întâlnim cu biblioteca virtuală, acum ajunsă la o nouă versiune, prezentând o interfață de portal prin intermediul căreia utilizatorii au acces la diverse resurse multimedia. Acesta este pretextul de care se folosește *Bogdan Manolache* pentru a discuta despre portalurile Web, unul dintre tipurile de situri care au o largă dezvoltare în ultimii ani, și – particularizând – despre portalurile educaționale. De asemenea, sunt prezentate și detalii privitoare la o parte din aspectele legate de implementarea în PHP a sitului dezvoltat și se dau unele amănunte referitoare la noua versiune PHP 5.

Universul PHP este prezent și în cel de-al patrulea capitol, dar, de această dată, în legătură cu modelarea relațiilor dintre tabelele bazelor de date MySQL, în contextul implementării unei aplicații aliniate problematicilor Web-ului semantic. Dintre aspectele captivante ale capitolului conceput de *Sorin Bahmătă* pot fi enumerate: descrierea conceptului de integritate referențială, prezentarea tipului de tabele *InnoDB*, introducerea cititorului în Web-ul semantic și prezentarea unora dintre tehnologiile pe care se bazează (de exemplu, metadate sau *RDF – Resource Description Framework*).

Tematica, extrem de interesantă, este continuată în următorul capitol, având-l ca autor pe *Cătălin Bulancea.* Aici se detaliază maniera de realizare a unui sit care permite vizualizarea unor baze de date relaționale (tabelele și relațiile dintre ele, plus informații despre câmpuri și înregistrări) prin intermediul limbajului *SVG* (*Scalable Vector Graphics*). Punctul central al capitolului îl reprezintă descrierea standardului SVG și a modului de utilizare a acestuia în contextul interacțiunii cu utilizatorul, pentru dezvoltarea efectivă a aplicației recurgându-se și la PHP și JavaScript.

Al șaselea capitol aduce în prim-plan o altă dimensiune de actualitate a spațiului WWW – *e-business*-ul. *Viorel Atofani* își propune să prezinte o soluție PHP de implementare a unui portal imobiliar și să discute cele mai

importante aspecte în demararea unei afaceri *on-line*. Se insistă mai ales asupra manierei de administrare direct pe Web a aplicației, iar din punctul de vedere al implementării se oferă o modalitate de procesare a fișierelor în format *CSV* (*Comma Separated Values*).

Următorul capitol migrează de la PHP la mai complexul și versatilul Perl, materialul punând la dispoziție detalii privitoare la maniera de implementare a unui sistem de management al *Weblog*-urilor. *Sabin Buraga* descrie, după o prezentare a fenomenului mondial al *Webloggíng*-ului, arhitectura conceptuală a aplicației și expune soluțiile de implementare a programelor constituente, recurgând la utilizarea expresiilor regulate Perl și la funcționalitățile modulului CGI. De asemenea, capitolul detaliază o soluție de folosire a șabloanelor de prezentare pentru a oferi vizitatorilor o interfață Web configurabilă.

Daniel Puiu și *Marian Bostan*, semnatarii capitolului opt, prezintă o modalitate de simulare a unui meci de fotbal dintre două echipe virtuale, implementând o suită de module ASP.NET. Cititorii vor putea studia în detaliu atât rezolvarea adoptată pentru simularea partidelor, cât și arhitectura aplicației, încorporată într-o soluție Visual Studio .NET complexă. Se descriu, printre altele, modul de realizare a *WebForms*-urilor, procesarea în limbajul C# a documentelor XML și utilizarea instrumentului SQLXML de generare a serviciilor Web pe baza procedurilor stocate în cadrul Microsoft SQL Server.

Cel din urmă capitol se concentrează asupra modului de creare a unor aplicații pentru dispozitivele mobile, astfel încât acestea să comunice cu platforma *desktop* cât mai eficient. De fapt, tema principală expusă de *Adrian Lazariuc* se referă la anatomia unei aplicații Palm OS – un catalog electronic –, prezentându-se principalele caracteristici ale manierei de programare pe dispozitive *handheld*. La final, se ilustrează posibilitatea utilizării practice a unui emulator Palm OS pentru testarea aplicației implementate. Partea *desktop* a aplicației realizate este concretizată într-un sit conceput în ASP.NET.

Lucrarea se încheie cu două anexe – una dedicată unei comparații între formatele *ShockWave Flash* și *Scalable Vector Graphics*, iar cealaltă descriind principalele caracteristici ale PHP și ASP.NET –, urmate de o bogată listă de referințe bibliografice.

Destinatari

Prin conținutul variat și circumscris direcțiilor actuale de dezvoltare a Web-ului, considerăm că lucrarea de față reprezintă un instrument de lucru indispensabil actualilor și viitorilor specialiști IT, în general, și dezvoltatorilor Web, în special.

Dintr-un punct de vedere, cartea reprezintă un îndrumar util atât studenților de la facultățile cu profil informatic (mai ales în cadrul laboratoarelor unor discipline precum *Tehnologii Web*, *Proiectarea siturilor Web*, *Sisteme distribuite*, *E-business*, *Tehnologii de elaborare a proiectelor software* sau *Ingineria programării*), cât și specialiștilor din domeniu (designeri, programatori, analiști, Webmasteri, arhitecți de baze de date și situri Web, administratori și alte categorii de personal din cadrul companiilor). De asemenea, anumite capitole pot prezenta interes și pentru profesorii și elevii de liceu.

Din alt punct de vedere, materialul demonstrează nivelul de cunoștințe pe care le poate acumula un tânăr, proaspăt absolvent de facultate, și edifică asupra unora dintre orizonturile reprezentative de dezvoltare a Web-ului asupra cărora ar putea/trebui să se concentreze în viitorul apropiat.

Mulțumiri

Coordonatorul volumului de față și al seriei *Web* mulțumește fiecărui autor de capitol pentru talentul, tenacitatea și verva investite atât în momentele de analiză, de programare sau de testare a aplicațiilor prezentate, cât și pe parcursul travaliului (poate uneori mai dificil decât scrierea efectivă a codului-sursă) de a explica și sintetiza ceea ce a realizat.

Autorii sunt recunoscători familiilor, prietenilor și colegilor pentru suportul, criticile constructive, înțelegerea și încurajările exprimate de atâtea ori de-a lungul timpului.

De asemenea, exprimăm gratitudinea noastră întregii echipe de profesioniști ai Editurii Polirom, fără de care această carte nu ar fi putut apărea.

Contact

Încurajând cititorii să experimenteze „magia" Web, îi invităm să ia contact cu fiecare dintre autorii capitolelor materialului de față și să viziteze situl dedicat cărții la *http://www.infoiasi.ro/~busaco/books/webapps/*.

Coordonatorul materialului de față poate fi contactat pe Web la adresa *http://www.infoiasi.ro/~busaco/*.

Sabin Buraga,
coordonatorul seriei *Web*
Iași, 20 septembrie 2004

CAPITOLUL I

România în Flash

Andrei Croitoru, icynoangel@yahoo.com

O scurtă descriere a mediului Flash MX și a limbajului ActionScript, reprezentarea unui șablon de prelucrare a fișierelor XML folosind această tehnologie, cu exemplificare concretă prin prisma a două module constituente ale aplicației *România în Flash*, vor reuși (sperăm) să vă convingă de ușurința și „libertatea de exprimare" în ceea ce privește programarea și design-ul Web, utilizând doar produsul oferit de Macromedia – Flash MX.

1. Introducere

Flash, ajuns deja la a șaptea versiune (pachetul Flash MX 2004), nu poate fi încadrat într-o singură categorie de medii de dezvoltare. Chiar și în urma enumerării calităților forte ale acestei aplicații, cititorul nu va rămâne poate uimit decât în momentul în care va deschide programul și va observa că informațiile existente pe Web pot lua foarte ușor o formă nouă, mai plăcută, sub „condeiul" propriei imaginații, utilizând un mediu de dezvoltare care, după părerea noastră, încearcă să adune ce este mai bun, mai folositor și mai estetic din mediile de dezvoltare existente.

Am putea spune că subiectul discuției noastre se va apropia mai mult de latura de prezentare a mediului Flash MX, a șasea versiune oferită de Macromedia, și de modalitatea de punere în practică a unei simple idei. Prin proiectul *România în Flash* s-a urmărit realizarea atât a unei prezentări generale a României, cât și a uneia zonale, pe județe, încercând totodată menținerea utilizatorului în sfera utilității aplicației prin desenarea dinamică a drumurilor naționale și europene de pe teritoriul fiecărui județ și oferindu-se posibilitatea determinării traseului minim dintre două localități.

Pe parcursul acestui capitol va fi prezentată o modalitate de realizare a unei galerii de imagini și o modalitate de desenare vectorială folosind ca mediu de dezvoltare Flash MX (ActionScript 1.0). Ca mijloace de stocare a datelor vom recurge la fișiere native *XML* (*Extensible Markup Language*).

2. Prezentare generală a mediului Flash MX

2.1 Introducere

Vom realiza mai întâi o familiarizare cu mediul Flash MX; o prezentare sumară a interfeței, a modurilor de stocare și afișare a informațiilor și, nu în ultimul rând, o vedere de ansamblu asupra limbajului de programare de tip script, asemănător cu JavaScript, și anume ActionScript.

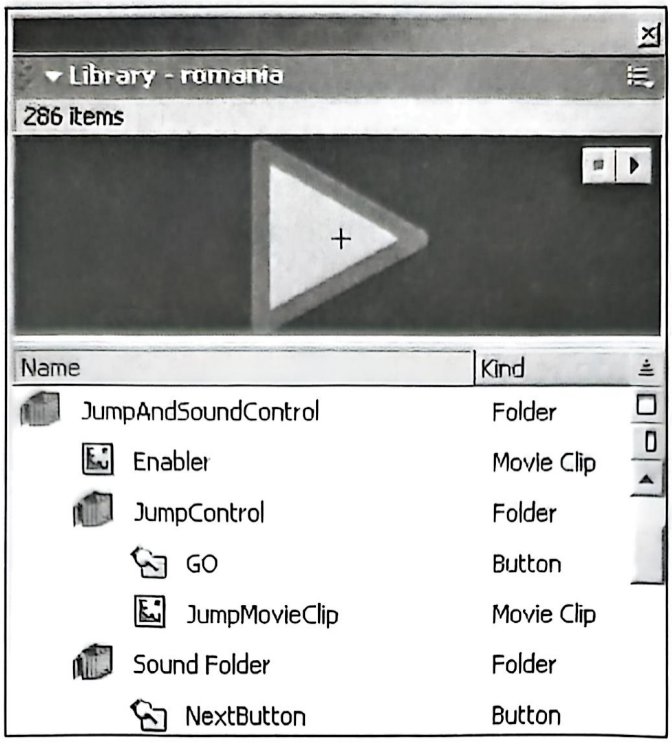
Flash MX este o aplicație care la prima vedere pare a fi un hibrid între un agent de prelucrare a imaginilor și un program de creare a graficii vectoriale, precum Macromedia Freehand sau Adobe Illustrator. Dar Flash este mult mai mult decât un astfel de hibrid. Flash reprezintă un program interactiv de prelucrare a fișierelor multimedia, un program sofisticat de creare de animații, de la ornamente Web simple până la emisii de conținut multimedia de calitate pentru Web (*broadcast*). Și, ca și cum atât nu ar fi suficient, Flash constituie gazda unui limbaj de programare de tip script, limbajul ActionScript, cu rădăcini în JavaScript (de fapt în ECMAScript, care este standardizat). Acest limbaj oferă posibilitatea de îmbinare între Flash și XML, fiind capabil să comunice cu alte componente ale Web-ului, cum ar fi: scripturi *PHP* (*PHP: Hypertext Preprocessor*), *ASP* (*Active Server Pages*), *ColdFusion* etc.

2.2 Prezentarea interfeței

Când spunem interfață a mediului Flash, vom încerca să ne limităm la a prezenta ferestrele existente și funcționalitatea acestora, fără a ne strădui să realizăm o lucrare de referință exhaustivă în sprijinul celor care vor dori să descopere acest mediu. Pentru o descriere detaliată, puteți studia cartea C. Vârlan, *Macromedia Flash*, Polirom, 2004.

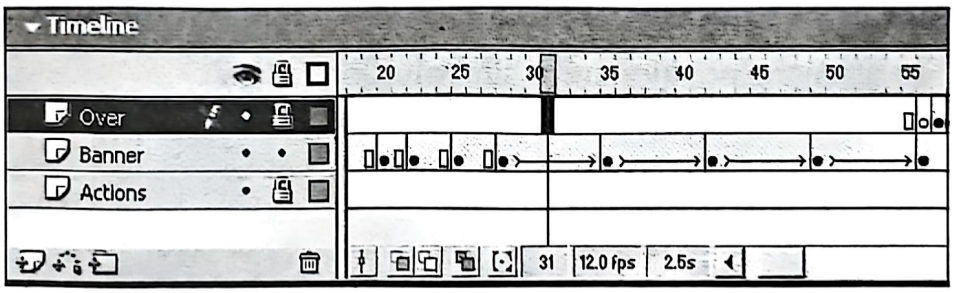
Vom stabili mai întâi câțiva termeni care vor fi folosiți foarte des pe parcursul acestui capitol, noțiunile fiind prezentate cu denumirile prin care sunt referite în documentația de specialitate.

- Prin *Stage* (scenă) vom înțelege locul unde se va desfășura acțiunea filmului, „pânza" de care vom „agăța" fie manual, fie prin intermediul codului ActionScript toate elementele ce vor apărea pe parcursul rulării aplicației.
- *Library* (bibliotecă) reprezintă zona unde avem depozitate toate elementele folosite în aplicație (elemente create manual cu ajutorul uneltelor de desenare sau importate în formate, dintre care amintim JPEG, PNG, AVI sau MPEG). Toate elementele stocate în bibliotecă poartă denumirea de *simboluri*.



Biblioteca mediului Flash MX (*Library*)

- Prin *Actions* (acțiuni) vom înțelege orice cod ActionScript.
- *Timeline* (linia timpului) reprezintă modul în care se succed evenimentele în Flash. Linia timpului ordonează conținutul Flash pe două dimensiuni: timp (maniera în care se succed cadrele – *frames*) și nivel – *layer* (modul în care sunt suprapuse informațiile).



Linia timpului (*Timeline*)

- Prin *Frame* (cadru) vom înțelege unitatea de măsură folosită pentru împărțirea liniei timpului.
- *Symbol* (simbol) reprezintă un element reutilizabil care se află în bibliotecă. De fiecare dată când vom plasa un simbol pe scenă sau în interiorul altui simbol, vom spune că lucrăm cu o *instanță* a acestui simbol. Spre deosebire de folosirea unui element grafic individual, utilizarea simbolurilor permite refolosirea unui anumit element fără a conduce la o creștere semnificativă în dimensiune a fișierului final. Simbolurile pot fi de trei tipuri:

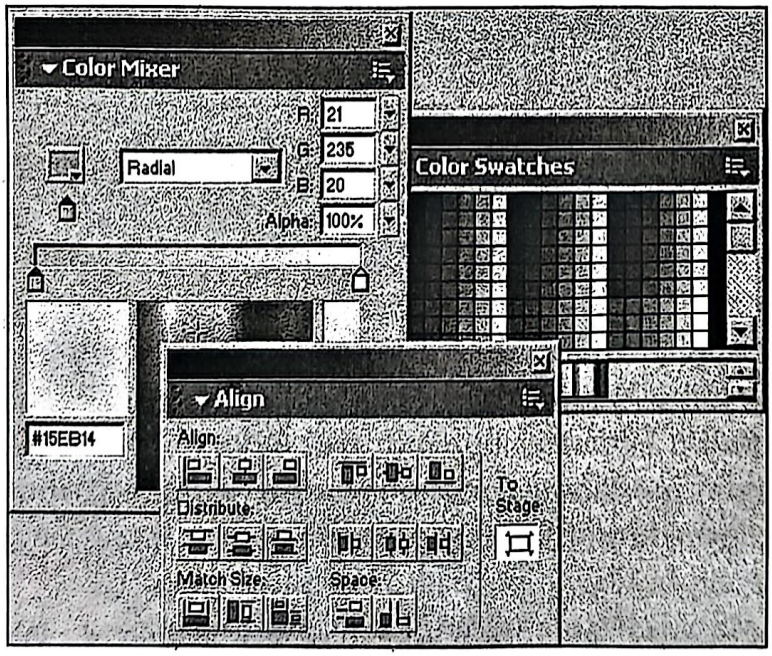
- simboluri de tip *Graphic* (grafic), folosite pentru imagini statice şi animaţii simple. Mediul Flash ignoră sunetele şi acţiunile din interiorul unor astfel de simboluri.
- simboluri de tip *MovieClip* (clip) sunt de tipul „*movie în movie*" (film în film). Se folosesc în general pentru animaţii care rulează independent de scena principală. Acestea pot conţine acţiuni, animaţii, sunete şi alte simboluri. Clipurile sunt utilizate foarte frecvent pentru crearea de butoane animate. Orice clip are o linie a timpului proprie.
- simboluri de tip *Button* (buton) sunt folosite pentru a crea butoane interactive. Butoanele au o linie temporală specială, cu patru cadre numite *stări – states*. Aceste stări sunt: *Up*, *Over*, *Down* şi *Hit*.

▪ Imaginile şi sunetele importate sunt şi ele tratate ca simboluri.

Una dintre cele mai utile zone ale interfeţei, mai ales în cazul în care am deschis mediul Flash pentru prima dată, o reprezintă *bara de unelte* (*Tools*). Aceasta este foarte asemănătoare cu bara de unelte din Adobe Photoshop, oferind posibilitatea de a crea interfaţa aplicaţiei dumneavoastră cu uşurinţa unui *click* de mouse.

Color Mixer şi *Color Swatches* reprezintă două ferestre foarte utile, permiţând selectarea culorii dorite şi crearea de efecte luminoase, realizând desenarea prin combinarea celor trei culori fundamentale – roşu, verde şi albastru (*RGB*) – cu un anumit grad de transparenţă (*alpha*).

O altă fereastră, fără de care aranjarea în pagină ar fi mult mai dificilă, este cea de aranjare/poziţionare a simbolurilor (*Align*).



Ferestrele funcţionale *Color Mixer*, *Color Swatches* şi *Align*

2.3 Prezentarea limbajului ActionScript 1.0

Limbajul ActionScript își are rădăcinile în JavaScript (ECMAScript) și a ajuns deja la versiunea 2.0, folosită o dată cu apariția pachetului Flash MX 2004. În realizarea acestei aplicații a fost utilizat mediul de dezvoltare Flash MX, care pune la dispoziție versiunea ActionScript 1.0.

Vom încerca o scurtă familiarizare cu acest limbaj, iar pe parcurs vom menționa o serie de amănunte referitoare la noua generație ActionScript 2.0.

Calea (*Path*)

Poate cel mai important aspect în mediul Flash îl reprezintă calea (*Path*) către simbolurile instanțiate. Să începem cu o noțiune simplă: scena este referită prin cuvântul-cheie `_root`. Pentru a specifica un simbol instanțiat pe scenă, se va folosi notația cu punct „.". De exemplu, pentru a referi un simbol numit `ball` (minge), aflat pe scenă, fragmentul de cod utilizat va fi: `_root.ball`.

Instanțierea

Să presupunem că în bibliotecă avem un simbol de tip *MovieClip* pe care îl instanțiem (lucru posibil atât prin ActionScript, cât și cu ajutorul mouse-ului). O instanțiere a unui simbol, fără a apela la cod, se realizează doar făcând *drag & drop* asupra simbolului din bibliotecă, poziționându-l pe scenă în locul dorit. Se va stabili pentru acesta un nume (*instance name*) prin care va fi referit. Instanțierea prin ActionScript se realizează în două etape:

- în primul rând se va seta pentru simbolul respectiv din bibliotecă un nume (*linkage name*). Acționați *click* dreapta pe simbol în bibliotecă, alegeți *Linkage Name*, apoi selectați *Export for ActionScript*, urmat de *Export in first frame.*
- pentru a „aduce" acest simbol pe scenă se va scrie următorul cod-sursă:

```
_root.createEmptyMovieClip("ball", 0);
_root.ball.attachMovie("Ball", "myBall", 0);
```

Să explicăm puțin codul de mai sus. Prima linie creează pe scenă un clip vid, numit `ball`, pe nivelul (*depth*) 0 de desenare. Linia a doua atașează clipului `ball`, creat anterior, simbolul `Ball` din bibliotecă, stabilește pentru acesta un nou nume, `myBall`, și îl așază pe nivelul 0 de desenare din interiorul clipului `ball`.

Vom referi acum simbolul dorit astfel: `_root.ball.myBall`.

Proprietăți

În momentul în care avem instanțiate simboluri, putem să modificăm anumite proprietăți ale acestora:

- `_x` și `_y` – poziția pe scenă sau în interiorul altui simbol (valoarea 0 pentru *x* și *y* este considerată colțul din stânga sus);
- `_alpha` – gradul de transparență al simbolului;
- `_height` și `_width` – înălțimea și lățimea simbolului;
- `_xscale` și `_yscale` – scara la care este redimensionat simbolul pe axele *x*, respectiv *y* (în procente);
- `_visible` – având două valori, `true` și `false`, stabilește dacă simbolul este vizibil sau nu;
- `_name` – reprezintă numele instanței simbolului.

Acțiuni – controlul clipurilor

Acțiunile pot fi scrise atât pentru cadre, cât și pentru simboluri. Strategia adoptată în această aplicație și sugerată în același timp de mai multe lucrări de specialitate este aceea de a avea atât pentru scenă, cât și pentru clipuri (acolo unde este nevoie) anumite straturi (*layers*), să le numim indispensabile, care să contribuie la o mai bună organizare, și, bineînțeles, la eficiență în depanarea sau îmbunătățirea aplicației. Iată care sunt acestea:

- *Actions* – strat care va reține toate acțiunile pentru cadre, pentru a facilita găsirea codului sursă ActionScript.
- *Labels* – va conține etichete pentru cadre; cu ajutorul etichetelor se va realiza saltul de la un cadru la altul.
- *Comments* – strat care va reține comentariile legate de anumite cadre.

Vom aplica această divizare logică referindu-ne la exemplul anterior. Mai întâi să precizăm câteva acțiuni pentru controlul clipurilor:

- `gotoAndPlay(frame)` – execuția clipului va „sări" la cadrul specificat și își va continua execuția; *frame* poate fi un număr sau o etichetă (*label*);
- `gotoAndStop(frame)` – execuția clipului va „sări" la cadrul specificat și se va opri acolo;
- `nextFrame()` – salt la cadrul următor;
- `prevFrame()` – salt la cadrul anterior;
- `play(), stop()` – pornește clipul mutând capul de citire la cadrele următoare, respectiv oprește clipul la cadrul curent;

Să presupunem că în exemplul nostru, în care atașăm un clip la scenă, codul corespunzător acestei acțiuni este scris în cadrul 3 al stratului *Actions*. Cadrului 3 din stratul *Labels* îi dăm numele `attach`. În primul cadru din

stratul *Actions*, adăugăm următoarea linie de cod: `stop();`. Execuţia filmului se va opri la primul cadru.

Pentru a ataşa clipul `ball`, este esenţial să ajungem la cadrul 3, unde avem codul ce realizează acest lucru. Pentru aceasta, vom scrie în primul cadru al stratului *Actions* următoarea acţiune: `gotoAndStop(3);` sau `gotoAndStop("attach");`.

Acţiuni – *Browser/Network*

Una dintre cele mai folosite acţiuni din această categorie este reprezentată de `loadMovie("url", level/target [, variables])`. Această acţiune încarcă în clipul curent (pe un anumit nivel) sau într-un alt clip existent (`level` sau `target`) un fişier Flash publicat pe Web, în format *.swf*, sau o imagine în format *.jpeg*, având calea `url`. În cazul în care este necesar să se transmită acestui clip anumite variabile, se va preciza modul în care valorile acestora vor fi transferate: `GET` sau `POST`.

```
on(release) {
   loadMovie("products.swf", _root.dropZone);
}
```

Acţiunea contrară este `unloadMovie()`, care înlătură clipul sau imaginea încărcată.

```
on (press) {
   unloadMovie ("_root.draggable");
   loadMovieNum ("movie.swf", 4);
}
```

Observaţie: în momentul în care se doreşte ca un clip să fie încărcat pe un nivel, `loadMovie()` se transformă în `loadMovieNum()`, acţiunea contrară fiind `unloadMovieNum()`.

O altă acţiune din această categorie este `getURL()`, care încarcă un document de la o anumită adresă într-o fereastră a *browser*-ului Web.

```
on(release) {
   getURL("http://www.infoiasi.ro", "_blank");
}
```

Variabile

Acţiunile pentru variabile sunt: `var`, `set`, `delete` şi `with`.

- `var variableName1 [= value1][...,variableNameN [=valueN]]` – creează una sau mai multe variabile și îi/le atribuie o valoare;
- `set(variable, expression)` – atribuie unei variabile valoarea expresiei `expression`;
- `delete reference` – distruge obiectul sau variabila reprezentată de `reference`;
- `with` – permite specificarea unui obiect și executarea anumitor acțiuni cu acesta, fără a scrie din nou numele obiectului.

```
with (someMovieClip) {
   _x = 50;
   _y = 100;
   gotoAndStop(3);
}
```

În acest exemplu, pentru clipul `someMovieClip` se modifică poziția acestuia pe axele *x* și *y*, după care, în interiorul său, se sare la cadrul 3, și execuția clipului se oprește la acest cadru.

Clase

Programarea orientată pe obiect și, în același timp, bazată pe evenimente este un punct forte al mediului Flash MX.

Vom preciza mai întâi câteva lucruri esențiale ale programării în ActionScript 1.0:

- variabilelor, în momentul declarării, nu le este precizat tipul. O variabilă își cunoaște tipul în momentul în care i se atribuie o valoare. Variabilele își pot schimba tipul în orice moment al execuției printr-o inițializare cu o valoare de alt tip (în ActionScript 2.0 se poate specifica tipul unei variabile în momentul în care aceasta este definită; astfel, variabila respectivă nu va mai putea să-și schimbe tipul și va genera o eroare în cazul atribuirii unei valori de alt tip).
- clasele nu sunt declarate în mod explicit, folosind cuvântul-cheie `class`, așa cum se întâmplă în cazul limbajului de programare C++ sau al noii versiuni ActionScript 2.0. Definirea claselor se realizează cu ajutorul cuvântului-cheie `function`. ActionScript 1.0 este un limbaj bazat pe prototipuri (`prototype`), ca și ECMAScript, din care derivă.

O clasă este constituită dintr-un constructor și un obiect-prototip. Obiectul-prototip este locul unde interpretorul Flash caută toate metodele și variabilele necesare execuției. Constructorul joacă rolul de șablon (*template*) pentru obiectele create în program. Iată un exemplu în care creăm o clasă simplă și instanțiem un obiect din această clasă:

```
MyCustomClass = function(){
   this.name = "customclass";
   this.description = "this is my first class";
}
customClassInstance = new MyCustomClass();
```

Prototipuri

Un prototip este un obiect, parte a tuturor claselor în Flash, care reţine toate metodele pentru clasa corespunzătoare. Deci prin prototip înţelegem locul unde sunt reţinute toate datele membre şi toate metodele pentru o clasă şi pentru obiectele instanţiate din clasa respectivă. Metodele unei clase nu vor fi copiate în fiecare instanţă a acesteia, utilizând astfel o zonă foarte mare de memorie, ci vor fi reţinute în prototipuri.

Moştenire

Vom explica acest concept printr-un exemplu:

```
SuperClass = function(){
   this._xscale = 10;
   this._yscale = 20;
};

SuperClass.prototype = new MovieClip;
SuperClass.prototype.onMouseDown = function(){
   this.mouseDetected();
} ;
SuperClass.prototype.mouseDetected = function(){
   trace(this._name + " a aparut un eveniment mouse");
};

SubClass = function(){
   this._x = this.x;
   this._y = this.y;
   super();
};

SubClass.prototype = new SuperClass;
Object.registerClass("square", SubClass);
```

În acest exemplu este creată mai întâi o clasă numită *SuperClass*, pe care o vom considera clasa-părinte, definită de utilizator. La momentul instanţierii acestei clase, se apelează constructorul clasei şi obiectului instanţiat i se vor atribui valorile 10, respectiv 20 pentru proprietăţile `_xscale` şi `_yscale`.

Prin linia de cod `SuperClass.prototype = new MovieClip;` stabilim faptul că orice obiect instanțiat din clasa *SuperClass* va moșteni toate funcțiile specifice clipurilor (`MovieClip`). Dacă este apelată o metodă care nu se găsește în prototipul clasei *SuperClass*, atunci interpretorul Flash va căuta această metodă în prototipul clasei `MovieClip`. În concluzie, prin această linie de cod se stabilește lanțul moștenirii între clasa creată de noi, *SuperClass*, și o clasă predefinită, `MovieClip`.

La pasul următor, se stabilește pentru clasa *SuperClass* o funcție membră care se va executa la apariția evenimentului `onMouseDown`. În corpul acestei funcții se afișează în fereastra de verificare (care apare doar în cazul testării aplicației în mediul de dezvoltare Flash) numele obiectului de unde a fost apelată metoda și un text scris de utilizator: `a aparut un eveniment mouse`.

În ceea ce privește clasa *SubClass*, stabilim să fie o subclasă a clasei *SuperClass* prin linia de cod: `SubClass.prototype = new SuperClass`. În momentul în care instanțiem un obiect din clasa *SubClass*, ar trebui să fie executat constructorul clasei-părinte, setându-se astfel anumite proprietăți și variabile. Pentru ca acest lucru să se întâmple, este necesar a se apela în mod explicit constructorul clasei-părinte cu ajutorul metodei `super()`.

`Object.registerClass()` este o metodă predefinită care atașează clasa definită unui obiect de tip `MovieClip`.

Observație: Definirea metodelor pentru fiecare clasă se va face după stabilirea clasei-părinte, deoarece acest lucru suprascrie prototipul subclasei.

3. Descrierea aplicației

Vom realiza o prezentare succintă a modulelor proiectului *România în Flash*, urmând să intrăm în detalii pentru două dintre acestea, și anume *Gallery* și *RouteMap.*

Primul modul și cel mai important îl constituie *modulul principal*, numit *Romania.* În cadrul acestuia sunt încărcate toate celelalte componente ale aplicației, realizându-se și comunicarea dintre acestea. Variabilele declarate în cadrul acestui modul pe nivelul 0 devin „variabile de nivel 0" pentru fiecare modul încărcat. Vom explica în continuare ce înseamnă acest lucru. Fiecare modul are un nivel 0 al său, acesta fiind nivelul său `_root`; fără a fi încărcat în modulul principal, orice referire la nivelul `_root` se referă la nivelul `_root` propriu. În momentul în care un modul este încărcat în cel principal, toate construcțiile folosind `_root` vor referi nivelul 0 al acestuia din urmă. De aceea, toate referirile din cadrul modulelor încărcate în modulul

principal utilizează referinţe relative, şi nu absolute, ceea ce înseamnă că în cadrul fiecărui modul toate construcţiile `_root` au fost înlocuite cu `this`. Putem considera aceste variabile, aflate pe nivelul 0 în modulul principal, ca fiind variabile globale, fără a mai fi nevoiţi să le declarăm globale via cuvântul-cheie `_global`. În această manieră, variabilele aflate pe nivelul 0 în modulul principal pot fi referite din orice modul încărcat în modulul principal, folosind cuvântul cheie `_root`. În concluzie, modulul principal joacă, în mare parte, rol de container pentru celelalte. Pe lângă acest fapt, modulul principal conţine şi anumite submodule de configurare vizuală (în acest caz, prin submodul înţelegem subrutine şi clipuri conţinute în modulul principal, utilizate la modificarea aspectului modulelor încărcate în modulul principal).

Modulul XMLMenu reprezintă meniul principal al aplicaţiei, utilizatorii putând selecta domeniul din care doresc să afle informaţii. Acest meniu este generat în mod dinamic folosindu-se un fişier XML în care sunt reţinute informaţiile necesare construirii meniului.

Modulul RomaniaIn este unul dintre modulele reprezentative ale aplicaţiei. În cadrul acestui modul sunt încărcate şi animate judeţele României (clipuri) şi reşedinţele de judeţ corespunzătoare. În acelaşi timp, pentru fiecare dintre acestea, pe baza unui fişier XML, sunt încărcate informaţii adiţionale, cum ar fi suprafaţa şi numărul de locuitori.

Modulul Generic prezentare constituie o clasă de module reprezentând codomeniul unei funcţii bijective, care are ca domeniu de definiţie opţiunile din meniul principal (a se vedea modulul *XMLMenu*).

Modulul Gallery este un modul în care sunt încărcate în manieră dinamică imagini din România pe baza unui fişier XML.

Modulul RouteMap este responsabil cu generarea drumurilor naţionale şi internaţionale de pe teritoriul României, pe baza datelor încărcate din fişierele XML care joacă rol de baze de date.

Modulul JumpTo este cel cu ajutorul căruia se pot obţine informaţii despre un anumit judeţ prin intermediul unei liste actualizate dinamic; acest modul are scopul de a oferi posibilitatea regăsirii informaţiei dorite într-un timp cât mai scurt.

Modulul Sound oferă vizitatorului posibilitatea de a selecta melodia de fundal care va rula pe timpul vizualizării sitului.

Partea de *intro* a sitului, *preloaderele* necesare încărcării fiecărui modul, *bannerele*, precum şi uneltele de configurare din cadrul modulului principal reprezintă module secundare ale aplicaţiei, având rolul de a atrage utilizatorul şi de a menţine legătura cu acesta şi atunci când o anumită parte a sitului se încarcă.

4. Detalii de implementare

4.1 Şablonul de prelucrare a fişierelor XML

Prelucrarea fişierelor XML este posibilă începând cu versiunea Flash 5, fiecare nouă versiune oferind metode suplimentare de acces la datele conţinute de fişiere externe.

Şablonul pe care îl vom prezenta este compus din două funcţii.

Prima dintre ele, numită `init()`, va crea un nou obiect de tip XML ce va reţine conţinutul fişierului XML sub forma unui arbore. Analiza fişierului XML fiind de tip *DOM – Document Object Model* (fişierul este încărcat în totalitate în memorie înainte de a fi prelucrat) –, se va stabili o funcţie ce va fi executată în momentul în care fişierul este încărcat complet.

Funcţia `processFile()` se va executa în momentul în care fişierul este încărcat în totalitate. Aceasta va conţine rutinele necesare extragerii informaţiilor din fişierul XML şi va fi modificată corespunzător cerinţelor problemei respective (pentru detalii, a se vedea secţiunile de prezentare a modulelor *Gallery* şi *RouteMap*).

```
function init(fileName) {
   //crearea obiectului XML
   my_xml = new XML();
   //ignorarea spatiilor albe din interiorul elementelor
   my_xml.ignoreWhite = true;
   //stabilirea functiei de prelucrare a fisierului
   my_xml.onLoad = function(success) {
      //incarcare completa - prelucreaza fisierul XML
      //continut de obiectul my_xml
      if (success) {
         processFile(my_xml);
      }
   };
    //incarcarea fisierului XML
   my_xml.load(fileName);
}
function processFile(my_xml) {
   //functia de prelucrare a fisierului XML
}
```

Analiza fişierului se va realiza prin proprietăţi specifice obiectelor de tip XML. Vom face o scurtă descriere a proprietăţilor celor mai des folosite pentru atingerea acestui obiectiv:

- `firstChild` – referinţă la primul nod din lista nodurilor fii ale nodului curent;

- `nextSibling` – referință la nodul frate al nodului curent, nod aflat pe același nivel cu nodul curent;
- `parentNode` – oferă o referință la nodul părinte;
- `childNodes` – returnează un vector conținând toate nodurile fii;
- `attributes` – returnează un vector asociativ al tuturor atributelor nodului respectiv;
- `nodeName` – returnează numele nodului curent;
- `nodeType` – returnează tipul nodului curent (1 – element, 3 – nod de tip text);
- `nodeValue` – returnează valoarea nodului curent;
- `loaded` – returnează `true` dacă fișierul XML este încărcat complet.

Observație: pentru obiectele de tip XML sunt oferite și metode de prelucrare a informațiilor conținute în nodurile arborelui construit prin procesarea fișierului XML: `appendChild`, `cloneNode`, `createElement`, `createTextNode`, `hasChildNodes`, `removeNode`.

4.2 Prezentarea modulului *Gallery*

Acest modul reprezintă o galerie virtuală, imaginile fiind încărcate în mod dinamic pe baza unui fișier XML în care sunt reținute pentru fiecare imagine calea unde se află stocată și o descriere pentru aceasta.

Pentru început, trebuie stabilit modul de organizare a fișierului XML reținând datele referitoare la imaginile ce vor fi încărcate în galeria noastră. Iată structura aleasă:

```
<?xml version="1.0" ?>
<pictures>
   <pic picPath = "RoInfo/Tourism/GalleriesPictures/
                  BlackSea/AquariumConstanta.jpg">
                  Aquarium Constanta
   </pic>
   <pic picPath = "RoInfo/Tourism/GalleriesPictures/
                  BlackSea/PlanetariumConstanta.jpg">
                  Planetarium Constanta
   </pic>
     ...
</pictures>
```

Folosind șablonul prezentat anterior, vom extrage informațiile necesare prezentării imaginilor din cadrul secțiunii „Marea Neagră" (*Black Sea*).

```
function init() {
   //modul de aparitie a imaginilor
   howToReveal = 0;
   //vector asociativ ce va retine datele din fisierul XML
   picturesArray = new Array();
   //crearea obiectului XML
   this.my_xml = new XML();
   //ignorarea spatiilor albe din interiorul elementelor
   my_xml.ignoreWhite = true;
   //stabilirea functiei de prelucrare a fisierului
   my_xml.onLoad = function(success) {
      if (success) {
         processFile(my_xml);
      }
   };
   //incarcarea fisierului
   my_xml.load(_root.nameOfGallery);
}

function processFile(my_xml) {
   //determinarea numarului de imagini
   this.numImages = my_xml.firstChild.childNodes.length;
   for (var i = 0; i < numImages; i++) {
      //salvarea proprietatilor pentru fiecare imagine
      picturesArray[i + 1] = new Array();
      //calea unde este salvata imaginea
      picturesArray[i + 1]["picPath"] =
      my_xml.firstChild.childNodes[i].attributes.picPath;
      //descrierea pentru imagine
      picturesArray[i + 1]["desc"] =
      my_xml.firstChild.childNodes[i].firstChild.nodeValue;
   }
}
```

Se observă că funcția `init()` nu mai primește în acest caz numele fișierului ca parametru, ci folosește pentru determinarea acestuia o variabilă (`nameOfGallery`) aflată pe nivelul 0 al modulului principal. Această variabilă este modificată în funcție de modulul care solicită încărcarea componentei *Gallery*.

În acest moment, având informațiile necesare salvate în vectorul asociativ `picturesArray`, putem începe încărcarea imaginilor.

```
function loadNewPic() {
   //contor ce va afisa numarul de ordine al imaginii
   this.picNumOutput = "";
```

```
    //variabila necesara afisarii
    //descrierii corespunzatoare imaginii
    this.descClip.descIn.desc_display = "";
    //modul de aparitie a imaginilor
    if(this.howToReveal < 2) {
       this.howToReveal++;
    } else {
       this.howToReveal = 0;
    }
    //atasarea clipului de mascare a imaginii
    this.attachMovie("photoreveal" + this.howToReveal,
                     "photoreveal1", 2);
    //pozitionare
    this.photoreveal1._x = 15;
    this.photoreveal1._y = 50;
    //incarcarea imaginii
    this.holder.loadMovie(picturesArray[picNum].picPath);
    //preincarcarea imaginii
    this.preloader.myPreloader.gotoAndStop(2);
}
```

Funcţia `loadNewPic()` are rolul de a încărca imaginea următoare din cadrul vectorului în care au fost salvate informaţiile din fişierul XML. Clipul `myPreloader` are misiunea de a preîncărca imaginea înainte de a o afişa în cadrul galerii. Tot în cadrul acestui clip se schimbă caracteristicile interfeţei cu utilizatorul, se actualizează numărul de ordine al imaginii, precum şi descrierea acesteia conform informaţiilor obţinute prin prelucrarea fişierului XML. Prezentăm în continuare secvenţa de cod din cadrul *preloader*-ului care realizează aceste acţiuni:

```
...
//rularea clipului de mascare a imaginii
_parent._parent.photoreveal1.gotoAndPlay(2);
//modificarea numarului de ordine a imaginii
_parent._parent.picNumOutput = _parent._parent.picNum +
   " of " + _parent._parent.numImages;

//stabilirea formatului pentru textul descrierii
_parent._parent.myTextFormat =
new TextFormat("MonotypeCorsiva", 17, 0xFFFFFF, true,
true);

//crearea clipului de descriere
_parent._parent.descClip.createEmptyMovieClip("descIn",0);
```

```
//crearea textului dinamic pentru afisarea descrierii
_parent._parent.descClip.descIn.createTextField(
  "textField", 100, 0, 0, 300, 80);
with (_parent._parent.descClip.descIn) {
   //stabilirea proprietatilor text
   textField.selectable = false;
   textField.align = "center";
   textField.embedFonts = true;
   textField.text =

_parent._parent.picturesArray[_parent._parent.picNum].desc
   //aplicarea formatarii textului
   textField.setTextFormat(_parent._parent.myTextFormat);
}
//realizarea aparitiei treptate a descrierii
_parent._parent.descClip.descIn._alpha = 0;
_parent._parent.descClip.descIn.onEnterFrame = function()
{
   if(this._alpha < 100) {
      this._alpha += 5;
   } else {
      delete this.onEnterFrame;
   }
}
...
```

Galeria de imagini

4.3 Prezentarea modulului *RouteMap*

În cadrul acestui modul sunt desenate drumurile naţionale şi internaţionale din cadrul fiecărui judeţ, oferindu-se posibilitatea de a afla traseul de lungime minimă dintre două oraşe din judeţul respectiv, folosindu-se algoritmul lui Dijkstra de aflare a drumului de cost minim într-un graf neorientat.

Prezentăm, pentru început, modul de organizare a fişierului XML pe baza căruia sunt generate drumurile în cadrul acestui modul, fişier care deţine în acelaşi timp informaţii ce fac posibil calculul drumului minim dintre două oraşe din judeţul respectiv.

```
<?xml version="1.0" ?>
<drawing>
  <!-- elementul 'outline' contine coordonate necesare desenarii
conturului judetului; idee: se deseneaza cate o curba pentru
fiecare 3-uplu epoint, vpoint, epoint, folosind functia curveTo
care necesita 3 parametri: punctul de start, punctul intermediar
si punctul final -->
  <outline>
     <epoint x="129" y="114" />
     <vpoint x="112" y="113" />
     <epoint x="116" y="127" />
     ...
     <epoint x="170" y="112" />
     <vpoint x="165" y="106" />
     <epoint x="129" y="114" />
  </outline>
  <!-- elementul 'roads' este folosit pentru desenarea drumurilor,
dar si pentru retinerea informatiilor necesare calcularii drumului
minim intre doua orase din cadrul judetului -->
   <roads>
   <!-- atributul 'name' retine numele/numarul drumului -->
     <m name="N1">
      <!-- fiecare element 'epoint' contine:
      - cele 6 coordonate necesare desenarii unei curbe,
fiecare epoint reprezinta o portiune din drumul corespunzator
referit de 'name';
      - parametrul 'pg' reprezinta numarul asociat bucatii
respective de drum;
      - parametrul 'c' contine numarul partilor de drum
adiacente cu cea referita de 'pg' in cadrul aceluiasi tag;
      - parametrul 'km' reprezinta numarul de kilometri
corespunzatori partii de drum reprezentate de 'pg' -->
       <epoint x="136" y="146" vx="145" vy="143" ex="163"
               ey="132" pg="0" c="1" km="3" />
       <epoint x="163" y="132" vx="180" vy="131" ex="193"
               ey="128" pg="1" c="0 2" km="2" />
```

```
    <epoint x="193" y="128" vx="210" vy="131" ex="219"
                ey="136" pg="2" c="1 3" km="3" />
      ...
    </m>
  <m name="E1">...</m>
  <m name="E2">...</m>
  <m name="E3">...</m>
  <m name="N2">...</m>
  <m name="N3">...</m>
  </roads>
    <!-- elementul 'towns' contine informatii referitoare la
pozitia (parametrii 'x' si 'y'), dimensiunea (parametrul 'tsize')
si numele oraselor, precum si la acele parti de drum cu care
intra in contact orasul respectiv (parametrul 'pg')-->
  <towns>
    <epoint x="136" y="146" pg="0" tsize="1">Bacesti
    </epoint>
    <epoint x="163" y="132" pg="1 0" tsize="1">Dumesti
    </epoint>
    <epoint x="250" y="140" pg="4 3" tsize="2">Negresti
    </epoint>
    <epoint x="286" y="157" pg="6 5" tsize="1">Vulturesti
    </epoint>
    <epoint x="336.95" y="225" pg="9 8" tsize="1">
    Stefan cel mare</epoint>
    <epoint x="387.95" y="275" pg="24 18 17 10" size="3">
    Vaslui</epoint>
    <epoint x="200" y="282" pg="12 11" tsize="1">
    Dragomiresti</epoint>
     ...
  </towns>
</drawing>
```

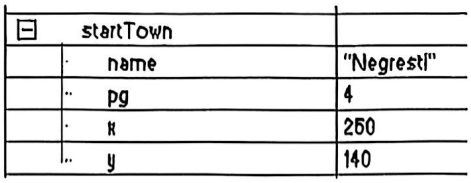
Funcţia `init()` folosită pentru analiza fişierului XML în acest caz este similară cu cea prezentată în şablonul descris mai sus. Prezentăm corpul funcţiei utilizate în cadrul acestui modul pentru prelucrarea conţinutului fişierului XML. Semnificaţia structurilor care apar în funcţia `processRFile()` va fi prezentată în paragrafele următoare.

```
function processRFile(myr_xml) {
  //determinarea datelor pentru desenarea conturului
  //judetului
  this.outline = myr_xml.firstChild.firstChild.childNodes;
  for (var i = 0; i < outline.length; i++) {
     oA[i] = new Array();
     oA[i]["x"] = outline[i].attributes.x;
     oA[i]["y"] = outline[i].attributes.y;
```

```
//determinarea datelor necesare desenarii soselelor
this.roads =
  myr_xml.firstChild.firstChild.nextSibling.childNodes;
for (var i = 0; i < roads.length; i++) {
   rnA[i] = roads[i].attributes.name;
   mA[i] = new Array();
   mA[i]["name"] = rnA[i];
   road = roads[i].childNodes;
   for (var j = 0; j < road.length; j++) {
      mA[i][j] = new Array();
      mA[i][j]["x"] = parseInt(road[j].attributes.x);
      mA[i][j]["y"] = parseInt(road[j].attributes.y);
      mA[i][j]["vx"] = parseInt(road[j].attributes.vx);
      mA[i][j]["vy"] = parseInt(road[j].attributes.vy);
      mA[i][j]["ex"] = parseInt(road[j].attributes.ex);
      mA[i][j]["ey"] = parseInt(road[j].attributes.ey);
      mA[i][j]["pg"] = parseInt(road[j].attributes.pg);
      mA[i][j]["km"] = parseInt(road[j].attributes.km);
      mA[i][j]["c"] = new Array();
      mA[i][j].c = road[j].attributes.c.split(" ");
   }
}
//determinarea datelor necesare pozitionarii oraselor
this.towns = myr_xml.firstChild.firstChild.nextSibling.
   nextSibling.childNodes;
for (var i = 0; i < towns.length; i++) {
   tA[i] = new Array();
   tA[i]["x"] = parseInt(towns[i].attributes.x);
   tA[i]["y"] = parseInt(towns[i].attributes.y);
   tA[i]["pg"] = new Array();
   tA[i].pg = towns[i].attributes.pg.split(" ");
   tA[i]["tsize"] = parseInt(towns[i].attributes.tsize);
   tA[i]["name"] = towns[i].firstChild.nodeValue;
}
}
```

Pentru a facilita înțelegerea codului sursă prezentat în continuare, vom oferi mai întâi o imagine de ansamblu asupra structurilor de date folosite – inițializările prezentate sunt conținute în funcțiile `init()` și `initGraf()`:

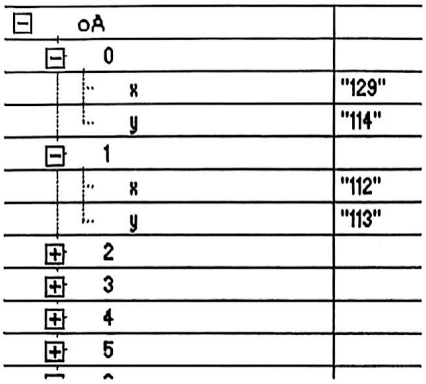
- `this.townSelected = 0;` – folosită pentru a reține câte orașe au fost selectate (0, 1 sau 2 – se declanșează procesul de calculare a drumului minim);
- `this.startTown = new Object();` – va conține informații referitoare la orașul de start, conform fișierului XML încărcat;



⊟ startTown	
name	"Negresti"
pg	4
x	250
y	140

Formatul vectorului asociativ `startTown`

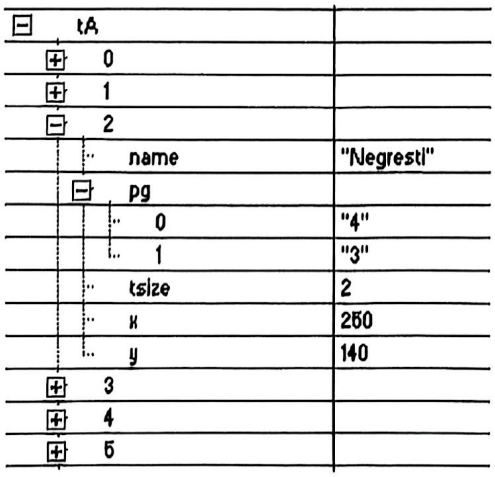
- `this.endTown = new Object();` – va conține informații referitoare la orașul destinație;
- `this.townTextFormat = new TextFormat("Arial", 14, 0xFFFFFF, true);` – formatarea necesară desenării numelor orașelor;
- `this.oA = new Array();` – structură folosită pentru a reține informațiile necesare desenării conturului județului;



⊟ oA	
⊟ 0	
x	"129"
y	"114"
⊟ 1	
x	"112"
y	"113"
⊞ 2	
⊞ 3	
⊞ 4	
⊞ 5	

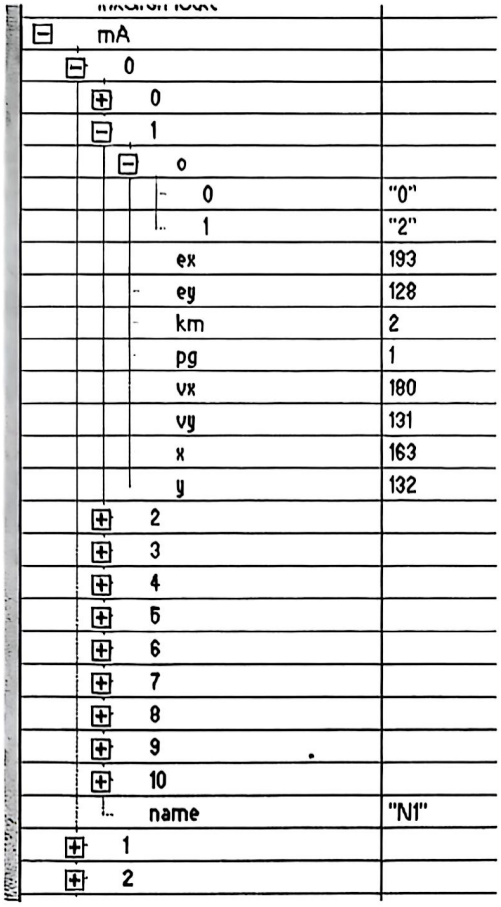
Formatul vectorului asociativ oA

- `this.mA = new Array();` – structură folosită pentru a memora informații legate de drumurile județului respectiv (semnificația fiind aceea precizată mai sus la prezentarea fișierului XML);
- `this.depthOfRoads = 10;` – nivelul (*depth*) de la care va începe desenarea drumurilor;
- `this.tA = new Array();` – structură folosită pentru a reține informații privitoare la orașe;



⊟ tA	
⊞ 0	
⊞ 1	
⊟ 2	
name	"Negresti"
⊟ pg	
0	"4"
1	"3"
tsize	2
x	250
y	140
⊞ 3	
⊞ 4	
⊞ 5	

Formatul vectorului asociativ tA



Formatul vectorului asociativ mA

- `this.depthOfTowns = 200;` – nivelul de la care se va începe ataşarea oraşelor (oraşele sunt simboluri de tip *Button*, salvate în bibliotecă);
- `this.gA = new Array();` – matricea care reţine graful necesar calculării drumului minim între două oraşe; stabileşte dacă două părţi de drum se intersectează sau nu (valori de 0 – nu există intersecţie, 1 – există intersecţie); nodurile în acest graf sunt reprezentate de părţi de lungime minimă de drum, iar muchiile stabilesc care dintre acestea se intersectează;
- `this.pred = new Array();` – vector ce reţine nodul anterior pentru fiecare nod;
- `this.cost = new Array();` – stochează costul fiecărei părţi de drum;
- `this.nodeScan = new Array();` – structură folosită pentru a memora nodurile în momentul parcurgerii grafului;
- `this.visited = new Array();` – vector ce reţine dacă un nod a fost vizitat sau nu;
- `this.theRoute = new Array();` – structură care va reţine o succesiune de noduri, constituind drumul minim dintre cele două oraşe selectate.

Vom prezenta în continuare:

- funcția care desenează conturul imaginar al județului – `drawOutline();`
- funcția ce desenează drumurile în cadrul județului – `drawRoads()`,
- prototipul `roadTrace()` folosit pentru afișarea unor etichete (*tooltips*), conținând denumirea șoselelor, la apariția evenimentului `onRollOver` asupra acestora.

```
function drawOutline() {
  //crearea clipului outlineMC
  this.createEmptyMovieClip("outlineMC", 0);
  //setarea stilului liniei
  this.outlineMC.lineStyle(5, 0x00CC33, 50);
  var i = 0;
  var ii = 2;
  //desenarea conturului
  this.outlineMC.moveTo(oA[i].x, oA[i].y);
  while (ii < outline.length) {
     this.outlineMC.curveTo(oA[i + 1].x, oA[i + 1].y,
     oA[ii].x, oA[ii].y);
     i = ii;
     ii += 2;
  }
  //afisarea etichetei la aparitia evenimentului onRollOver
  outlineMC.onRollOver = function() {
     //crearea clipului "tooltip"
     this._parent.createEmptyMovieClip("toolTip",
       toolTipDepth);
     //atasarea clipului "ToolTip" din biblioteca
     this._parent.toolTip.attachMovie("ToolTip",
       "innerToolTip", 0);
     //setarea textului pentru eticheta
     this._parent.toolTip.innerToolTip.tip_display
        = "County border";
     with(this._parent) {
        toolTip._x = _xmouse + 5;
        toolTip._y = _ymouse - 20;
     }
     //setarea etichetei sa urmareasca mouse-ul
     this._parent.toolTip.startDrag(false, 0, 0, 900, 750);
  };
  //eliminarea etichetei la aparitia evenimentului onRollOut
  outlineMC.onRollOut = function() {
     this._parent.toolTip.removeMovieClip();
  };
}
```

```
function drawRoads() {
  for (var j = 0; j < mA.length; j++) {
     auxName = mA[j].name;
     //crearea clipurilor pentru desenarea soselelor
     this.createEmptyMovieClip("road" + auxName,
       depthOfRoads);
     depthOfRoads++;
     //setarea culoarii in functie de tipul soselei:
      //drum european sau national
     if (auxName.indexOf("E") != -1) {
        this["road" + auxName].lineStyle(3, 0xCC0000, 100);
     } else if (auxName.indexOf("N") != -1) {
        this["road" + auxName].lineStyle(3, 0xFFFF00, 100);
     }
     //desenare in interiorul clipului corespunzator soselei
     for (var i = 0; i < mA[j].length; i++) {
        this["road" + auxName].moveTo(mA[j][i].x,
           mA[j][i].y);
        this["road" + auxName].curveTo(mA[j][i].vx,
          mA[j][i].vy, mA[j][i].ex, mA[j][i].ey);
     }
  }
  //atasarea de actiuni clipurilor ce reprezinta soselele
  for (var j = 0; j < mA.length; j++) {
     auxName = mA[j].name;
     this["road" + auxName].roadTrace();
  }
}

//prototip folosit pentru afisarea numelui soselei in
//momentul aparitiei evenimentului onRollOver
MovieClip.prototype.roadTrace = function() {
  this.onRollOver = function() {
     //crearea unui clip vid
     this._parent.createEmptyMovieClip("toolTip",
       toolTipDepth);
     //atasarea clipului ToolTipRoads din biblioteca
     this._parent.toolTip.attachMovie("ToolTipRoads",
       "innerToolTip", 0);
     //extragerea numelui soselei din numele clipului
     //reprezentand soseaua respectiva
     tipAuxName = this._name.substr(4);
     //stabilirea textului ce va fi afisat
     this._parent.toolTip.innerToolTip.tip_display =
       tipAuxName;
     //pozitionarea etichetei in zona in care se afla mouse-ul
```

```
    with(this._parent) {
        toolTip._x = _xmouse + 5;
        toolTip._y = _ymouse - 20;
    }
    //setarea etichetei sa urmareasca mouse-ul
    this._parent.toolTip.startDrag(false, 0, 0, 900, 750);
    };
  this.onRollOut = function() {
    //eliminarea clipului eticheta
    this._parent.toolTip.removeMovieClip();
  };
};
```

Înainte de a prezenta funcţiile folosite pentru calcularea drumului minim între două oraşe, vom explica modul în care se realizează desenarea acestui drum folosind funcţia `travel()` – în momentul în care drumul minim a fost deja calculat – şi maniera în care lucrează algoritmul amintit.

Funcţia `travel()` este folosită pentru desenarea unei porţiuni de drum, mai precis desenarea unui <epoint> din interiorul unui element <m> din fişierul XML de intrare. Desenarea traseului între două oraşe se realizează apelându-se această funcţie pentru fiecare porţiune de drum, porţiune componentă a rutei determinate prin algoritmul lui Dijkstra. Redarea grafică se realizează treptat, folosind `onEnterFrame`.

```
function travel(roadPg, depthOfTravel) {
  //crearea unui clip nou pentru a desena pe un nivel superior
  //celui pe care sunt desenate soselele
  this.createEmptyMovieClip("travel" + depthOfTravel,
    depthOfTravel);
  //pozitionare in mA
  breakFor = false;
  for (var i = 0; i < mA.length; i++) {
    for (var j = 0; j < mA[i].length; j++) {
      if (mA[i][j].pg == roadPg) {
        breakFor = true;
        break;
      }
    }
    if (breakFor) {
      break;
    }
  }
  //desenarea portiunii de drum
  this["travel" + depthOfTravel].lineStyle(5,
    0xCC66FF, 100);
```

```
    this["travel" + depthOfTravel].moveTo(mA[i][j].x,
      mA[i][j].y);
    this["travel" + depthOfTravel].curveTo(mA[i][j].vx,
      mA[i][j].vy, mA[i][j].ex, mA[i][j].ey);
}
```

Foarte succint, vom prezenta modul în care este determinat drumul minim într-un graf neorientat folosind algoritmul lui Dijkstra (pentru o prezentare detaliată a acestuia, se pot consulta lucrările de specialitate din domeniul algoritmicii). Pentru reprezentarea grafului am utilizat o matrice pătratică având dimensiunea egală cu numărul de porțiuni de drum existente în cadrul județului respectiv. Nodurile (mulțimea *V*) sunt reprezentate de aceste porțiuni de drum (o porțiune de drum corespunde unui element `<epoint>` din cadrul unui element `<m>` al fișierului XML respectiv), muchiile (mulțimea *M*) stabilesc dacă două porțiuni de drum se intersectează sau nu. Vom prezenta modul de funcționare a algoritmului prin prisma implementării realizate.

Matricea pătratică *gA* reține graful necesar calculării drumului minim (prin valori de 1 se stabilește faptul că porțiunile de drum corespunzătoare liniei și coloanei respective se intersectează).

Vectorul *visited* este folosit pentru a reține dacă un nod a fost vizitat sau nu (*true* sau *false*);

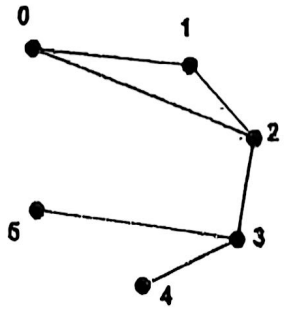
Vectorul *pred* reține pentru fiecare nod al grafului nodul din care s-a ajuns la nodul curent în momentul parcurgerii grafului.

Vectorul *cost* stochează costul corespunzător fiecărei porțiuni de drum, respectiv al fiecărei muchii a grafului.

Vectorul *nodeScan* este utilizat pentru a memora nodurile grafului în ordinea în care acestea sunt vizitate.

Modul de funcționare a algoritmului

Să considerăm un graf simplu cu 6 noduri și 6 muchii.



Reprezentare grafică

	0	1	2	3	4	5
0	0	1	1	0	0	0
1	1	0	1	0	0	0
2	1	1	0	1	0	0
3	0	0	1	0	1	1
4	0	0	0	1	0	0
5	0	0	0	1	0	0

Matricea corespunzătoare

Vectorii *pred, cost, visited* au dimensiunea 6. *Pred* are inițial valoarea –1 pe fiecare poziție, *cost* – valoarea 0 pe fiecare poziție, *visited* – valoarea *false* pe fiecare poziție. Să presupunem că dorim să calculăm drumul minim de la nodul 4 la nodul 0. Vectorii se vor modifica astfel:

Pred						Cost						Visited						Node Scan
0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5	4, 3
-1	-1	-1	4	-1	-1	0	0	0	0	0	0	f	f	f	f	t	f	
0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5	4, 3, 2, 5
-1	-1	3	4	-1	3	0	0	0	1	0	0	f	f	f	t	t	f	
0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5	4, 3, 2, 5,
2	2	3	4	-1	3	0	0	2	1	0	0	f	f	t	t	t	f	0, 1
0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5	4, 3, 2, 5,
2	2	3	4	-1	3	0	0	2	1	0	2	f	f	t	t	t	t	0, 1
0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5	4, 3, 2, 5,
2	2	3	4	-1	3	3	0	2	1	0	2	t	f	t	t	t	t	0, 1
0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5	4, 3, 2, 5,
2	2	3	4	-1	3	3	3	2	1	0	2	t	t	t	t	t	t	0, 1

Evoluția vectorilor *Pred*, *Cost*, *Visited* și *NodeScan*

În momentul în care se vizitează un nod, se vor adăuga în vectorul *nodeScan* nodurile adiacente care nu există deja în acest vector și se va marca faptul că nodul curent a fost vizitat (în vectorul *visited* se marchează cu `true` poziția corespunzătoare). În plus, se actualizează costul pentru nodul respectiv prin incrementarea costului nodului precedent – nodul care a determinat adăugarea nodului curent în vectorul *nodeScan* (în vectorul *pred* se vor reține nodurile pentru care nodul curent este predecesor). Pentru găsirea drumului, se va parcurge vectorul *pred*, modalitatea de determinare a drumului minim fiind următoarea: se pleacă de la nodul 0, nodul final în cazul considerat, și se continuă cu predecesorul acestuia (nodul 2 în cazul considerat). Acest lucru se execută până la întâlnirea valorii –1 (valoare corespunzătoare nodului de start). Drumul minim în acest caz este: 0, 2, 3, 4.

În cazul aplicației de față, algoritmul folosit este cel prezentat mai sus, însă în momentul parcurgerii grafului corespunzător, costul fiecărei muchii nu se mai obține prin incrementare la costul nodului precedent, ci este citit

din matricea *mA* ca fiind numărul de kilometri corespunzători porţiunii respective de drum, la care se adaugă costul nodului precedent (structura matricei *mA* a fost descrisă mai sus).

Prezentăm în continuare funcţiile implementate pentru determinarea drumului minim dintre două oraşe din cadrul unui judeţ.

```
//reunirea tuturor functiilor necesare si desenarea drumului
//utilizand functia „travel"
function calculateRoute() {
  //stergerea rutei determinate anterior
  this.clearB.enabled = false;
  //initializarea grafului
  initGraf();
  //crearea grafului
  createGraf();
  //initializarea vectorilor auxiliari
  initGrafRoute();
  /determinarea rutei
  createGrafRoute(startTown, endTown);
  //desenarea rutei determinate
  pOnRoute = 0;
  routeDepth = 60;
  this.onEnterFrame = function() {
     if (pOnRoute < theRoute.length) {
        //desenarea fiecarui segment in parte din ruta
        //determinata
        travel(theRoute[pOnRoute], routeDepth);
        pOnRoute++;
        routeDepth++;
     } else {
        this.clearB.enabled = true;
        delete this.onEnterFrame;
     }
  };
}

//crearea matricei grafului
function createGraf() {
//fiecare sosea va primi un numar pentru crearea matricei
//de intersectie, aceste numere sunt alocate pentru
//fiecare sosea in ordinea in care acestea apar in
//fisierul XML
//determinarea dimensiunii matricei
  this.dim = 0;
  for (var i = 0; i < mA.length; i++) {
     dim += mA[i].length;
  }
```

```
  //initializarea matricei cu valoare 0
  for (var i = 0; i < dim; i++) {
     gA[i] = new Array(dim);
     for (var j = 0; j < dim; j++) {
        gA[i][j] = 0;
     }
  }
}

//initializarea vectorilor pred, cost, visited
//modificarea matricei corespunzator intersectiilor
//dintre muchiile grafului
function initGrafRoute() {
  for (var i = 0; i < dim; i++) {
     this.pred[i] = 0;
     this.cost[i] = 0;
     this.visited[i] = false;
  }
  for (var i = 0; i < mA.length; i++) {
     for (var j = 0; j < mA[i].length; j++) {
        for (var k = 0; k < mA[i][j].c.length; k++) {
           gA[mA[i][j].pg][mA[i][j].c[k]] = 1;
        }
     }
  }
}

//determinarea drumului minim dintre orasul de start si
//orasul destinatie
//se vor exclude acele parti care nu trebuie desenate si
//implicit nu trebuie luate in considerare
function createGrafRoute(startTown, endTown) {
  //analiza grafului se face in ordine inversa
  pg1 = startTown.pg;
  pg2 = endTown.pg;
  //pozitia curenta in NodeScan
  pNS = 0;
  nodeScan[pNS] = pg2;
  pred[pg2] = -1;
  //determinarea costului pentru punctul de start (pg2)
  breakFor = false;
  for (var k = 0; k < mA.length; k++) {
     for (var kk = 0; kk < mA[k].length; kk++) {
        if (mA[k][kk].pg == pg2) {
           breakFor = true;
           break;
```

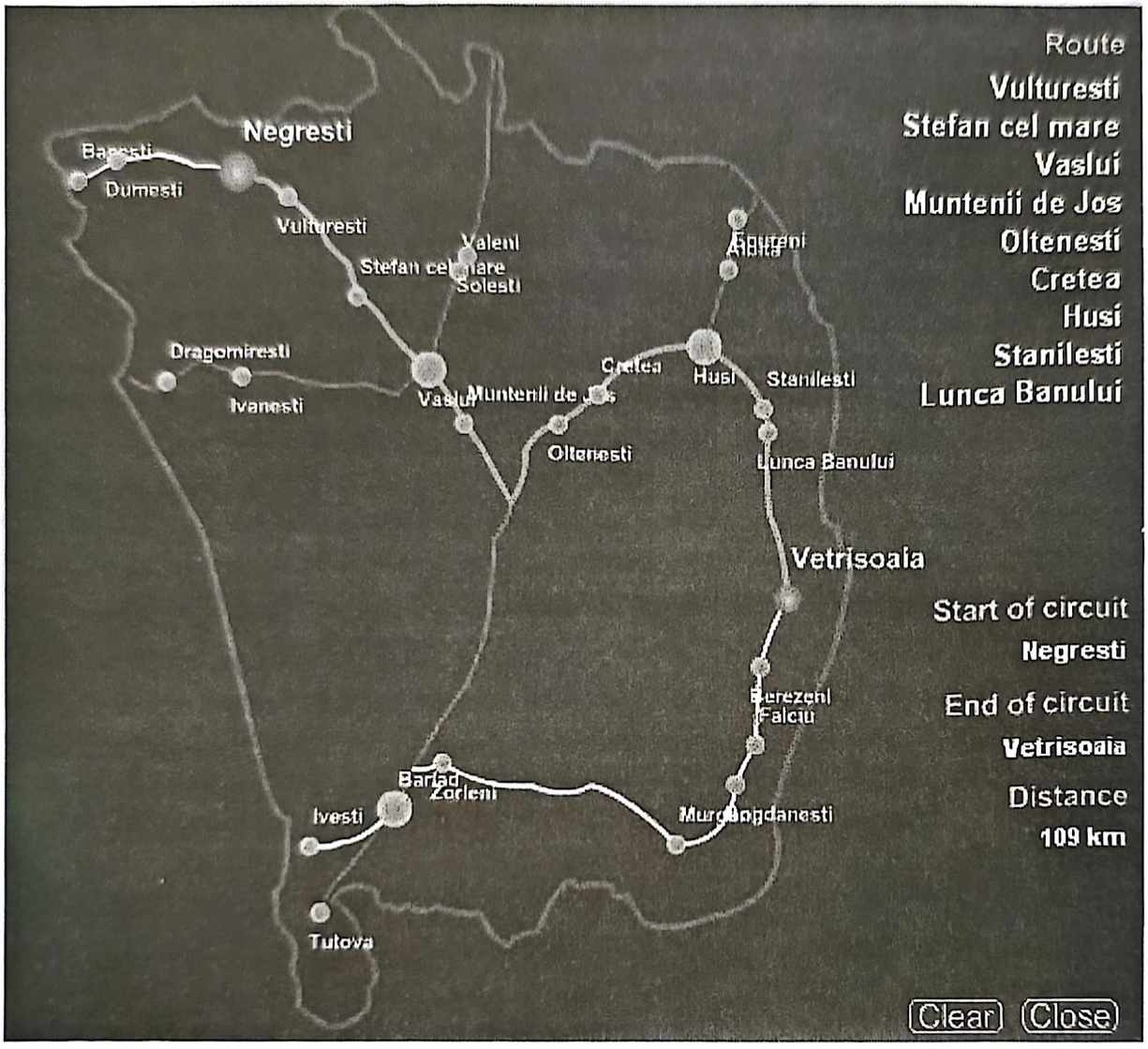
```
      }
   }
   if (breakFor) {
      break;
   }
}
cost[pg2] = mA[k][kk].km;
visited[pg2] = true;
//aplicarea algoritmului Dijkstra
while (pNS < visited.length) {
   for (var j = 0; j < visited.length; j++) {
      if (gA[nodeScan[pNS]][j] == 1) {
         if (!visited[j]) {
            nodeScan.push(j);
            pred[j] = nodeScan[pNS];
            breakFor = false;
            //determinarea pg corespunzator pentru
            //a adauga km
            for (var k = 0; k < mA.length; k++) {
            for (var kk = 0; kk < mA[k].length; kk++) {
                  if (mA[k][kk].pg == j) {
                     breakFor = true;
                     break;
                  }
               }
               if (breakFor) {
                  break;
               }
            }
            cost[j] = cost[nodeScan[pNS]] + mA[k][kk].km;
            visited[j] = true;
         }
      }
   }
   pNS++;
}

pPred = pg1;
theRoute.push(pg1);
//crearea drumului minim prin adaugarea nodurilor la ruta
while (pred[pPred] != -1) {
   theRoute.push(pred[pPred]);
   pPred = pred[pPred];
}
//stabilirea costului initial pentru ruta determinata
routeCost = cost[theRoute[0]];
```

```
  //pentru depasire in partea de start, se localizeaza
  //partea a doua a rutei determinate pentru a vedea daca
  //prima parte trebuie luata in considerare sau nu
  //pozitia in pred
  breakFor1 = false;
  for (var i = 0; i < mA.length; i++) {
     for (var j = 0; j < mA[i].length; j++) {
        if (mA[i][j].pg == theRoute[1]) {
           breakFor1 = true;
           break;
        }
     }
     if (breakFor1) {
        break;
     }
  }
//se testeaza daca orasul de pornire (startTown) se afla si
//pe a doua portiune din ruta determinata, caz in care prima
//portiune va fi eliminata din aceasta ruta
if ((startTown.x == mA[i][j].x && startTown.y == mA[i][j].y)
 || (startTown.x == mA[i][j].ex &&
      startTown.y == mA[i][j].ey)) {
     routeCost = cost[theRoute[1]];
     theRoute.reverse();
     theRoute.pop();
     theRoute.reverse();
  }
  //pentru depasire in partea finala,
  //se localizeaza penultima portiune a rutei determinate
  breakFor1 = false;
  for (var i = 0; i < mA.length; i++) {
     for (var j = 0; j < mA[i].length; j++) {
        if (mA[i][j].pg == theRoute[theRoute.length-2]) {
           breakFor1 = true;
           break;
        }
     }
     if (breakFor1) {
        break;
     }
  }
//se testeaza daca orasul destinatie (endTown) se afla in
//penultima portiune a rutei determinate, caz in care
//ultima portiune va fi eliminata din aceasta ruta
```

```
if ((endTown.x == mA[i][j].x && endTown.y == mA[i][j].y)
  || (endTown.x == mA[i][j].ex && endTown.y ==
mA[i][j].ey)) {
     routeCost -= cost[theRoute[theRoute.length-1]];
     theRoute.pop();
  }
  //partea de afisare a informatiilor suplimentare, cum
  //ar  fi: numarul de kilometri pentru ruta determinata
  //si al  oraselor intermediare
  this.distance_km = routeCost + " km";
  this.start_town = this.startTown.name;
  this.end_town = this.endTown.name;
  //afisarea oraselor intermediare
  route_display = "";
  for (var i = 0; i < theRoute.length; i++) {
     //j este folosit pentru localizarea numelui in tA
     for (var j = 0; j < tA.length; j++) {
        for (var k = 0; k < tA[j].pg.length; k++) {
           //se testeaza daca orasul curent se afla
           //situat pe portiunea curenta
           if (theRoute[i] == tA[j].pg[k]) {
           //se testeaza daca orasul curent se afla deja in
           //variabila de afisare si in acelasi timp nu
           //reprezinta orasul de start sau orasul
           //destinatie
              if (route_display.indexOf(tA[j].name) == -1
                    && (tA[j].name != startTown.name &&
                    tA[j].name != endTown.name)) {
                 //adaugarea orasului la variabila de afisare
                 route_display += tA[j].name + newline;
                 break;
              }
           }
        }
     }
   }
}
```

Înainte de a desena ruta determinată, se va verifica dacă prima și ultima porțiune trebuie desenate sau nu. Fiecare oraș are un indice (sau mai mulți), parametrul <pg> din fișierul XML. Primul indice reprezintă porțiunea de drum pentru care punctul de start corespunde cu coordonatele orașului. Revenind la ideea enunțată anterior, vom elimina din ruta determinată prima porțiune, în cazul în care coordonatele orașului de start coincid cu cele de start ale porțiunii a doua (ambele porțiuni au punctul de start în același loc –

orașul aparține și de prima, *pg* corespunde primei, și de a doua la începutul acesteia –, ceea ce ar fi imposibil) sau dacă coordonatele orașului corespund finalului celei de-a doua. Într-o formă mai explicită: dacă orașul de start aparține și celei de-a doua porțiuni din ruta calculată, atunci prima porțiune va fi eliminată din rută. În ceea ce privește ultima porțiune din ruta calculată, se verifică dacă orașul destinație aparține penultimei porțiuni (începutului sau sfârșitului acesteia), caz în care ultima parte din ruta calculată va fi eliminată. În ambele situații, costul total se modifică în mod corespunzător.



Desenarea drumurilor din județul Vaslui

5. Comentarii bibliografice

În acest capitol s-a încercat familiarizarea cititorului cu mediul Flash MX (ActionScript 1.0), ajungându-se spre final la un nivel avansat de programare care necesită o oarecare experiență în domeniul paradigmei obiectuale și, în

același timp, bazată pe evenimente. Acest lucru nu ar trebui să îi descurajeze însă pe cei care sunt la prima întâlnire cu Flash, mediul de dezvoltare oferindu-le posibilitatea de a-și pune în practică ideile actuale, precum și de a-și îmbunătăți tehnicile de dezvoltare de aplicații profesioniste pe măsura acumulării de experiență.

Un punct de pornire în descoperirea Flash-ului îl constituie cartea *Flash 5 Bible*, scrisă de Robert Reinhardt și Jon Warren Lentz și apărută în 2001 la John Wiley & Sons, carte care – deși detaliază o versiune mai veche a mediului de dezvoltare Flash – reprezintă o lucrare de referință pentru dezvoltarea de aplicații și în versiunile recente.

Pentru a realiza trecerea de la Flash 5 la Flash MX se poate consulta lucrarea *Flash MX Savvy*, apărută în 2004 la Sybex și având-i ca autori pe Ethan Watrall și pe Norbert Herber, cartea prezentând în amănunt modul în care poate fi utilizat Flash MX pentru dezvoltarea de aplicații profesioniste.

Pentru orice întrebare sau nelămurire, se pot accesa oricând siturile dedicate problematicilor ActionScript: *http://www.actionscript.org/* sau *http://www.flashkit.com/*. Aceste situri sunt o sursă inestimabilă de informații în ceea ce privește programarea în ActionScript și nu numai.

CAPITOLUL II

De la Flash la PHP. Dezvoltarea unui forum Web

Cosmin Vârlan, vcosmin@uaic.ro

Mai mult ca sigur, ca exploratori ai Internet-ului, v-ați întâlnit și ați discutat în cadrul diverselor forumuri. Probabil este timpul să învățați cum se construiește un forum, să studiați pe ce tehnologii se bazează; poate doar vă tentează să realizați o astfel de aplicație în Flash. Capitolul va prezenta maniera de interoperabilitate dintre Flash și serverul de aplicații PHP, precum și realizarea de pagini Web dinamice.

1. Descrierea forumului

Construirea unui forum în Flash nu este nici pe departe atât de dificilă pe cât v-ați fi închipuit. În exemplul următor vom parcurge pașii prin intermediul cărora puteți construi o astfel de aplicație. Deoarece scopul acestui capitol este de a vă învăța cum să realizați un forum, și nu de a prezenta PHP-ul sau Flash-ul[1], vom presupune că deja ați mai lucrat cu aceste tehnologii și aveți cunoștințe primare privitoare la acestea.

Pe partea de client, vom folosi *Macromedia Flash MX 2004*. Dintre avantajele acestei tehnologii menționăm: exportarea prezentărilor multimedia dinamice în fișiere de dimensiuni mici care pot fi descărcate rapid de pe Internet, posibilitatea de adăugare a unor interacțiuni complexe cu vizitatorul, efectele de animație, încărcarea de tip *stream* a datelor. Avantajul pe care îl oferă un forum în Flash este cel referitor la maniera de transmisie între server și client a informațiilor: dacă într-un forum de tip HTML trebuia ca – o dată cu informația utilă – să trimiteți un întreg set de reguli de așezare în pagină sau pentru tabelare (folosind, de exemplu, proprietăți CSS sau elemente HTML de prezentare), utilizând o interfață Flash, aceste reguli vor fi impuse la construcția aplicației ce va rula pe mașina clientului și nu veți avea de transmis decât datele utile (subiecte de discuție, mesaje redactate etc.).

1. Pentru a studia tehnologia Flash mai în amănunt, recomandăm cartea Cosmin Vârlan, *Macromedia Flash*, Editura Polirom, Iași, 2004 – *http://www.infoiasi.ro/~flash*.

Pe partea de server, vom folosi scripturi PHP versiunea 4, iar pentru a stoca diverse informații vom recurge la serverul de baze de date MySQL. Datele transmise între server și client vor fi structurate ca fișiere *XML* (*Extended Markup Language*), care permit o bună organizare a informațiilor.

2. Realizarea în PHP a aplicației care rulează pe server

Pentru început, ne vom concentra atenția asupra modului de funcționare a unui forum. Pașii pe care un vizitator trebuie să îi urmeze în utilizarea unui forum sunt:

- *înregistrarea* – pentru această operație vom avea nevoie de un formular în care vom solicita informații despre utilizator (nume de cont, parolă, adresă *e-mail*, adresă Web a sitului personal etc.);
- *posibilitatea de autentificare* – aici va trebui să luăm în considerare probleme cum ar fi securitatea;
- *introducerea unui nou subiect de discuție*;
- *adăugarea unui răspuns* la un subiect deja existent.

De asemenea, deși mai puțin importante, am putea avea unele secțiuni opționale, cum ar fi: vizualizarea listei tuturor membrilor înregistrați, listarea informațiilor privitoare la un anumit utilizator sau schimbarea preferințelor de interfață (opțiuni pe care le veți implementa dumneavoastră, după ce veți aprofunda materialul capitolului de față).

2.1 Formatul tabelelor din baza de date

Urmând pașii enumerați mai sus, putem intui o structură a unei baze de date – modul de organizare a informațiilor fiind una dintre problemele pe care trebuie să ni le punem la începerea unei astfel de aplicații.

Baza de date pe care o vom crea va purta numele `ForumDB` și va conține trei tabele: `utilizatori`, `subiecte` și `mesaje`. Cele trei tabele pot avea câmpuri diverse și pot fi modificate pentru a vă satisface propriile preferințe în dezvoltarea finală a aplicației; pentru exemplul implementat aici, cele trei tabele vor conține următoarele informații:

- *utilizatori* – stochează informațiile privitoare la un utilizator, având drept câmpuri principale pe cele date în continuare:
  - `IDutilizator` (identificator de tip *int [11]* atașat fiecărui utilizator, folosit drept cheie primară și având setată proprietatea de autoincrementare);
  - `utilizator` (*varchar [20]*);
  - `parola` (*varchar [40]*);
  - `nume` (*varchar [20]*);

- `prenume` (*varchar [40]*);
- `varsta` (*int [2]*);
- `sex` (*varchar [1]*);
- `localitate` (*varchar [30]*);
- `email` (*varchar [100]*);
- `web` (*varchar [255]*);
- `posturi` (*int [11]*);
- `avatar` (*varchar [20]*);

▪ *subiecte* – conține lista tuturor subiectelor, având în componență câmpurile:

- `IDsubiect` (*int [11], autoincrement*);
- `IDutilizator` (*int [11]* – identifică persoana care a deschis subiectul);
- `subiect` (*varchar [200]* – reprezintă titlul subiectului);
- `mesaje` (*int [11]*);
- `data` (*int [11]* – data ultimului mesaj);

▪ *mesaje* – memorează lista tuturor mesajelor redactate, indiferent de subiect:

- `IDmesaj` (*int [11], autoincrement*);
- `IDsubiect` (*int [11]* – pentru a realiza legătura cu un anumit subiect);
- `IDutilizator` (*int [11]* – reprezintă persoana care a scris mesajul);
- `mesaj` (*mediumtext* – mesajul propriu-zis);
- `data` (*int [11]* – data la care a fost adăugat).

Baza de date poate fi creată într-o manieră ușoară prin executarea unui script PHP al cărui cod sursă nu îl furnizăm aici. Încurajăm utilizatorul să consulte cartea S. Buraga (coord.), *Aplicații Web la cheie. Studii de caz implementate în PHP*, apărută în 2003 la Editura Polirom.

2.2 Programele PHP stocate pe server

Pe server va trebui să scriem diverse fișiere cod sursă PHP care să confere posibilitatea de transmitere a informațiilor către Flash, respectiv de preluare a informațiilor din Flash. Urmărind pentru a doua oară modul de funcționare a forumului, putem stabili ce funcționalități vor fi implementate de programele PHP dezvoltate.

Pentru preluarea informațiilor transmise de Flash și stocarea lor într-o bază de date, vom avea nevoie de fișierele:

- *inregistrare.php* (crearea unui nou cont);
- *subiectnou.php* (adăugarea unui subiect);
- *mesajnou.php* (adăugarea unui mesaj).

De asemenea, opțional, putem construi încă un program PHP care să modifice informațiile privitoare la utilizator (*modificautilizator.php*).

Pentru transmiterea informațiilor preluate din baza de date MySQL către Flash, ne vom folosi de fișierele:

- *vizualizeazasubiecte.php* (va returna un obiect XML cu lista tuturor subiectelor);
- *vizualizeazasubiect.php* (va returna un document XML conținând toate mesajele dintr-un subiect).

Atunci când vom dori să autentificăm un utilizator, ne vom folosi de *verificare.php* și, optional, vom putea concepe fișierul *vizualizeazautilizator.php* pentru a afla unele informații despre ceilalți utilizatori.

Funcțiile comune mai multor operații enumerate mai sus (de exemplu, funcția de conectare la baza de date sau cea de verificare a datelor transmise spre autentificare) vor fi incluse într-un alt fișier: *comun.php*. Toate fișierele PHP vor fi stocate într-un director cu numele *forumengine*; în directorul părinte vor fi adăugate *index.html*, *forum.swf* și încă un director numit *avatars*.

În continuare, vor fi listate parțial conținuturile fiecăruia dintre fișierele PHP. În cadrul codului, vor fi date și unele explicații pentru o mai bună înțelegere sau pentru a putea modifica diferiți parametri spre a particulariza codul astfel încât să convină nevoilor dumneavoastră.

Alături de funcția `Conectare()`, fișierul *comun.php* va conține:

```
<?
// autentificarea utilizatorului
function auth($utilizator, $parola) {
  $crypt = md5($parola);  // parola codificata cu MD5
  //formulam o cerere - numele si parola utilizatorului:
  $cerere = "SELECT IDutilizator FROM utilizatori WHERE
       utilizator = '$utilizator' AND parola =
       '$crypt'";
  $rezultat = mysql_query($cerere);

  if (mysql_num_rows($rezultat) == 1) {
    $user = mysql_fetch_array($rezultat);
    $IDutilizator = $user['IDutilizator'];
    // memoram ID-ul
  } else {
    $IDutilizator = -1; // dam ID-ului valoarea -1
  }
  return $IDutilizator;
  // este returnat ID-ul utilizatorului
}
```

```
// verifica daca un e-mail este valid
function checkEmail($email)
{
     // definirea unei expresii regulate
     $regexp = "^[_a-z0-9-]+(\.[_a-z0-9-]+)*@[a-z0-9-
]+(\.[a-z0-9-]+)*(\.[a-z]{2,4})$";
     return (eregi($regexp, $email));
}
?>
```

În fișierul *comun.php* s-au definit patru funcții:

`Conectare()` are rolul de a crea o nouă conexiune la baza de date; va returna fie o eroare reprezentând motivul pentru care PHP-ul nu a putut lua legătura cu MySQL-ul, fie valoarea 1, în cazul reușitei.

Funcția `fail()` are rolul de a forma un nou șir de tip URL care conține mesajul de eroare, după care va forța ieșirea din orice program PHP.

Funcția `auth()` va primi ca parametri un cont de utilizator și o parolă și va returna fie ID-ul utilizatorului, fie valoarea –1 în cazul în care asocierea dintre utilizator și parolă este incorectă (observați faptul că parametrul parolă este codificat prin algoritmul MD5 și este comparat cu șirul găsit în baza de date, deci în baza de date vor fi păstrate codificat toate parolele).

Ultima funcție – `checkEmail()` – are rolul de a verifica dacă adresa de *e-mail* este una validă. Pentru aceasta, ne folosim de expresii regulate în stilul Perl: adresa de *e-mail* poate începe doar cu caracterul „_", o literă de la „a" la „z" sau o cifră de la 0 la 9; despărțite de punct, pot urma grupări ale acelorași caractere, apoi este obligatoriu să fie găsit caracterul „@" urmat de cel puțin o grupare a aceluiași set de litere și, în final, de o extensie de două până la patru caractere (domeniul Internet).

Scriptul *inregistrare.php* va fi responsabil cu înregistrarea (crearea) unui utilizator al forumului. Înregistrarea unui nou utilizator are în vedere verificarea existenței unui utilizator cu același nume (de cont). În cazul în care nu s-a găsit o astfel de persoană, informațiile transmise vor fi introduse în baza de date. Lăsăm pe cititor să scrie acest program simplu.

Scriptul *subiectnou.php* primește din Flash variabilele `utilizator`, `parola`, `subiect`, `mesaj` și inserează în baza de date un nou subiect de discuții împreună cu un prim mesaj:

```
<?
include('comun.php');

// conectarea la baza de date
$con = Conectare();
```

```
// autentificarea utilizatorului
$IDutilizator = auth($utilizator, $parola);
if ($IDutilizator == -1) { // autentificare esuata
  fail("Utilizator inexistent sau parola incorecta!");
}

// in cazul in care autentificarea s-a efectuat cu
// succes:
$data = time(); // se preia ora transmiterii mesajului
// cerere pentru completarea listei subiectelor:
$cerere = "INSERT INTO subiecte (IDutilizator, subiect,
  data) VALUES ($IDutilizator, '$subiect', $data)";
if(!mysql_query($cerere)) { // se executa cererea
  fail("Eroare in inserarea subiectului");  // iesire
}
// impreuna cu subiectul transmitem primul mesaj:
$IDsubiect = mysql_insert_id();
// cerere pentru adaugarea noului mesaj:
$cerere = "INSERT INTO mesaje (IDsubiect, IDutilizator,
  mesaj, data) VALUES($IDsubiect, $IDutilizator,
'$mesaj', $data)";
if(!mysql_query($cerere)) {  // executarea cererii
  fail("Eroare in adaugarea mesajului");  // iesire
}
print "&rezultat=Okay";  // noul subiect a fost inserat
mysql_close($con);     // deconectare
?>
```

Poate vă întrebaţi din ce motiv trebuie să autentificăm un utilizator de fiecare dată când introducem un nou subiect sau mesaj. Răspunsul la această întrebare este destul de simplu: dacă nu am face acest lucru, probabil că un vizitator destul de „ingenios" ar putea trimite informaţii care să fie postate pe forum în numele altor persoane, eventual folosind un formular realizat în HTML. Nu dorim să se întâmple acest lucru.

Programul *mesajnou.php* preia de la Flash variabilele `utilizator`, `parola`, `IDsubiect`, `mesaj` şi inserează un mesaj în cadrul subiectului al cărui identificator a fost recepţionat:

```
<?
...
// crearea unei cereri pentru introducerea unui mesaj:
$cerere = "INSERT INTO mesaje (IDsubiect, IDutilizator,
  mesaj, data) VALUES($IDsubiect, $IDutilizator,
'$mesaj', $data)";
```

```
if(!mysql_query($cerere)) { // executarea cererii
  fail("Eroare in introducerea mesajului");
}
// crearea unei cereri pentru incrementarea numarului
// de mesaje dintr-un anumit subiect:
$cerere = "UPDATE subiecte SET mesaje = mesaje + 1,
                data = $data WHERE IDsubiect =
                $IDsubiect";
if(!mysql_query($cerere)) {
  fail("Eroare de conectare la lista subiectelor");
}
$cerere = "UPDATE utilizatori SET posturi = posturi + 1
                  WHERE IDutilizator = $IDutilizator";
if(!mysql_query($cerere)) {
  fail("Eroare de conectare la lista utilizatorilor");
}
print "&rezultat=Okay";
mysql_close($con);
?>
```

Şi de această dată trebuie să verificăm identitatea utilizatorului înainte de a introduce în baza de date mesajul trimis (secvenţa de verificare e similară celei de la scriptul precedent).

Programul *modificautilizator.php* primeşte ca parametri `utilizator`, `parola_curenta`, `parola`, `avatar`, `email`, `nume`, `prenume`, `varsta`, `sex`, `localitate`, `web` şi modifică aceste informaţii pentru utilizatorul cu numele conţinut de variabila `utilizator`:

```
<?
...
$crypt = md5($parola); // criptarea noii parole
if (!checkEmail($email)) { // verificarea adresei de
                           // mail
    fail("Adresa de mail invalida.");
}
// autentificarea utilizatorului pe baza parolei vechi:
if (auth($utilizator, $parola_curenta)==-1){
   fail ("Parola curenta nu a fost introdusa corect !");
}
$cerere = "UPDATE utilizatori SET parola='$crypt',
  avatar='$avatar', email='$email', nume='$nume',
  prenume='$prenume', varsta='$varsta', sex='$sex',
  localitate='$localitate', web='$web'
  WHERE IDutilizator = '$id' LIMIT 1";
```

```
$rezultat = @mysql_query($cerere);
print "rezultat=$rezultat";  // afisarea rezultatului
mysql_close($con);
?>
```

Fereastra care permite utilizatorului modificarea informațiilor privitoare la contul său (adresa de *e-mail*, pagina Web, numele etc.) va fi asemănătoare celei în care se creează un nou utilizator. În programul PHP va trebui să verificăm dacă utilizatorul este cu adevărat cine pretinde că este, și din acest motiv vom testa din nou dacă parola veche este corectă (parola veche în cazul în care a fost introdusă o nouă parolă). Dacă rezultatul acestei testări este pozitiv, se va trece la modificarea informațiilor vechi cu cele noi primite din Flash.

După cum observați, am construit toate fișierele PHP care duc la adăugarea sau modificarea înregistrărilor. Vom trece în continuare la scrierea scripturilor PHP care au rolul de a interoga baza de date și de a trimite rezultatul către aplicația Flash. Așa cum am precizat mai sus, acest transfer se va realiza prin structurarea informațiilor în obiecte XML.

Primul program este cel care va transmite către Flash lista tuturor subiectelor și se numește *vizualizeazasubiecte.php* (nu primește nici un parametru și returnează un document XML conținând lista subiectelor împreună cu câteva informații aferente – data redactării, autorul, numărul de răspunsuri):

```
<?
...
// se vor genera subiectele in ordine invers
// cronologica:
$cerere = "SELECT * FROM subiecte
           ORDER BY data DESC";
$rezultat = mysql_query($cerere); // executam cererea
if (!$rezultat) { // mesaj de eroare in caz de esec
    fail("Lista subiectelor nu poate fi listata!");
}
// reusita va conduce la numararea subiectelor:
$threadCount = mysql_num_rows($rezultat);
$output = "<?xml version=\"1.0\" ?>"; // construim un
                                      // document XML
$output .= "<threads>"; // adaugam un element
// dupa care listam pe rand toate subiectele:
for ($count = 0; $count < $threadCount; $count++)
{
  $thread = mysql_fetch_array($rezultat);
  $IDsubiect = $thread['IDsubiect']; // IDsubiect
  $IDutilizator = $thread['IDutilizator']; // IDutilizator
```

```
    $subiect = stripslashes($thread['subiect']); // subiect
    $raspunsuri = $thread['mesaje']; // raspunsuri
    $data = strftime("%d/%m/%y %H:%M",
                $thread['data']); // data ultimului mesaj
    // aflam numele utilizatorului care a scris subiectul:
    $cerere = "SELECT utilizator FROM utilizatori
               WHERE IDutilizator = $IDutilizator";
    $rezultat2 = @mysql_query($cerere);
    $user = @mysql_fetch_array($rezultat2);
    $utilizator = $user['utilizator'];
    // continuam construirea obiectului XML prin
    // adaugarea informatiilor aflate mai sus:
    $output .= "<thread id=\"".$IDsubiect."\"
subiect=\"".
                $subiect."\"
postat_de=\"".$utilizator."\"
                raspunsuri=\"".$raspunsuri."\" data=\"".
                $data."\" />";
}
$output .= "</threads>"; // inchiderea tag-ului XML
echo $output;            // afisarea XML-ului
mysql_close($con);
?>
```

Deşi modul de construire a obiectului XML este puţin artificial (pentru că nu ne-am folosit de funcţiile PHP specializate de generare a conţinutului XML), acesta va fi bine formatat şi va fi pe deplin înţeles de programul Flash. Acest fişier PHP are rolul de a transmite Flash-ului toate subiectele scrise (postate). Aţi putea modifica liniile de cod de mai sus, astfel încât transmiţând din Flash două variabile – care să indice intervalul dorit de subiecte (de exemplu, subiectele de la 30 la 40) – să fie returnate numai subiectele din acest interval. Astfel, puteţi obţine foarte uşor o paginare a subiectelor.

Scriptul *vizualizeazasubiect.php* va primi o variabilă reprezentând ID-ul subiectului ce trebuie afişat şi va returna un obiect XML conţinând toate mesajele, autorii şi datele la care au fost introduse:

```
<?
...
// cererea va returna toate mesajele dintr-un subiect
// ordonate ascendent dupa data postarii:
$cerere = "SELECT * FROM mesaje WHERE IDsubiect =
           $IDsubiect ORDER BY data ASC";
$rezultat = @mysql_query($cerere);
if (!$rezultat) {
    fail("Nu s-au gasit mesaje!");
}
```

```
// sau in cazul in care s-au gasit mesaje, sunt
// numarate:
$postCount = @mysql_num_rows($rezultat);
// este creat obiectul XML:
$output = "<?xml version=\"1.0\" ?>";
$output .= "<mesaje>";
for ($count = 0; $count < $postCount; $count++) {
    $post = mysql_fetch_array($rezultat);
    $IDutilizator = $post['IDutilizator'];
    $mesaj = stripslashes($post['mesaj']);
    $data = strftime("%d/%m/%y %H:%M", $post['data']);
    // cerere pentru aflarea numelui (stim ID-ul):
    $cerere = "SELECT utilizator, avatar, posturi
       FROM utilizatori
       WHERE IDutilizator = $IDutilizator";
    $rezultat2 = @mysql_query($cerere);
    $user = @mysql_fetch_array($rezultat2);
    $utilizator = $user['utilizator'];
    $avatar = $user['avatar'];
    $posturi = $user['posturi'];
    $output .= "<post autor=\"".$utilizator."\"
        data=\"".
        $data."\" avatar=\"".$avatar."\" posturi=\"".
        $posturi."\" IDutilizator=\"".$IDutilizator.
        "\" mesaj=\"".$mesaj."\"  />";
}
$output .= "</mesaje>";
echo $output;
mysql_close($con);
?>
```

Programul `verificare.php` primeşte numele şi parola unui utilizator şi returnează ID-ul acestuia sau valoarea –1 dacă utilizatorul nu există sau nu are setată parola dată ca parametru:

```
<?
include('comun.php');
$link = Conectare();
$IDutilizator = auth($utilizator, $parola);
print "rezultat=".$IDutilizator;
?>
```

Acest program PHP nu face decât să returneze ID-ul utilizatorului în cazul în care numele şi parola au fost corect introduse.

În cazul în care se vor solicita informaţiile referitoare la un anumit utilizator, Flash-ul va trebui să transmită scriptului PHP identificatorul utilizatorului.

Această informație va fi trimisă către *vizualizeazautilizator.php*, care va returna un document XML construit ca și în cazurile precedente – nu mai furnizăm codul sursă.

3. Realizarea în Flash a aplicației care rulează la client

În continuare, ne vom îndrepta atenția către aplicația Flash care va rula pe calculatorul clientului. Pentru dezvoltarea acesteia, se va folosi versiunea MX 2004 a programului Macromedia Flash.

Deschideți Macromedia Flash și, într-un nou document, setați dimensiunile scenei la 650 x 400 pixeli. Deoarece nu avem nevoie să animăm obiecte în scena principală, scriptul va avea doar un singur cadru (deși vor fi necesare mai multe straturi – *layers* – pentru afișarea diverselor elemente grafice).

Trebuie menționat faptul că toate *MovieClip*-urile care vor fi construite în Flash vor avea punctul de înregistrare în colțul din stânga sus (cu excepția celor în care se specifică altfel).

Pe primul strat veți construi un *MovieClip* care să conțină un dreptunghi de dimensiunile 650 x 50 pixeli, pe care îl veți așeza în partea de sus a forumului. În acest *MovieClip* veți scrie numele forumului (de exemplu, „Forumul maniacilor de Flash") sau veți putea încărca un SWF sau JPEG exterior.

Trebuie să restricționăm afișarea *form*-urilor sau a zonei în care se va face afișarea subiectelor/mesajelor. Din acest motiv, pe cel de-al doilea strat vom crea o mască cu dimensiunile 650 x 330 pixeli pe care o vom poziționa la 5 pixeli distanță față de *MovieClip*-ul aflat în primul strat. Convertiți dreptunghiul care constituie masca în *MovieClip* și dați-i numele de instanță `masca`.

Al treilea *layer* va fi ascuns de către masca desenată în precedentul. El va conține un *MovieClip* cu numele de instanță `gol` în care vom încărca diversele *form*-uri sau *MovieClip*-uri. Acest *MovieClip* nu va conține nici un element grafic și va fi poziționat exact în colțul din stânga sus al măștii. Pentru a-l masca prin intermediul lui `masca`, nu vom proceda așa cum se face de obicei când lucrăm cu grafică, ci vom realiza această mascare din interiorul *ActionScript*-ului (motivul fiind că lui `gol` va trebui să-i atașăm alte *MovieClip*-uri care, atunci când conțin text de tip *dynamic*, nu vor fi afișate corect).

Al patrulea strat va fi redenumit `ActionScript` și în interiorul său vom scrie toate acțiunile care trebuie să fie atașate liniei temporale principale.

Stratul cu numărul 5 este unul temporar (pentru a putea desena eventualele *form*-uri și a le raporta corect la dimensiunile scenei).

Pe ultimul strat (al şaselea) fie desenaţi un degradé (gradient) radial care să realizeze un efect de *fade* din partea stângă sus către dreapta jos, fie importaţi o imagine. Redenumiţi acest ultim strat `background`. Blocaţi toate straturile, cu excepţia celui temporar (5); rezultatul ar trebui să arate ca în figura următoare:

Şablonul de afişare a interfeţei forumului

3.1 Construirea modulelor de autentificare/înregistrare

Construirea Flash-ului va urma o ordine firească a funcţionalităţii: degeaba puteţi viziona subiectele, dacă nu aveţi nici un subiect sau mesaj introdus; la fel, pentru introducerea unui subiect nou veţi avea nevoie de măcar un utilizator înregistrat.

Atunci când sunt transmise cereri pentru adăugarea unui subiect nou sau a unui mesaj, trebuie să fim siguri că persoana care a făcut această cerere este chiar cine pretinde a fi. Din acest motiv e necesar ca în forumul nostru să includem un modul de autentificare.

Autentificarea utilizatorului este foarte simplă şi se face prin apelul programului *verifica.php*, având ca parametri numele şi parola utilizatorului. Dacă rezultatul verificării este pozitiv, utilizatorului i se va permite să adauge un subiect, un mesaj, să editeze informaţiile privitoare la persoană etc. Toate aceste opţiuni vor fi afişate într-un meniu care, atunci când nimeni nu este autentificat, va conţine o legătură către lista subiectelor (în cazul în care cineva vizualizează mesajele fără a fi autentificat, trebuie să aibă o posibilitate de întoarcere la lista subiectelor), respectiv una spre modulul de autentificare. Partea dreaptă a forumului a fost alocată special pentru a afişa aceste meniuri.

Creaţi un buton cu numele de instanţă `autentificare`, la apăsarea căruia veţi lansa în execuţie funcţia ActionScript `_root.autentifica()`. Selectaţi aşadar butonul şi ataşaţi-i următorul script:

```
on (release) {
     _root.autentifica();
}
```

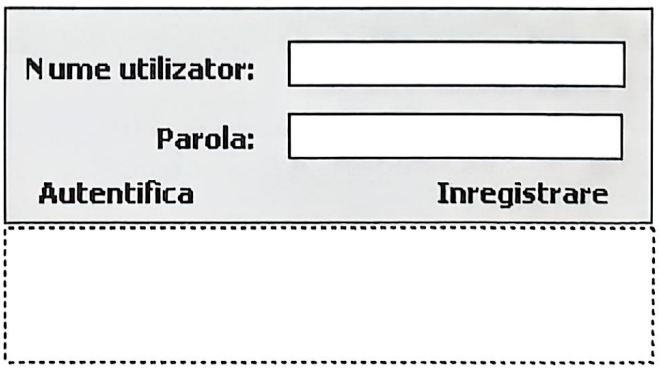
Adăugați și butonul de afișare a listei subiectelor („Afișează subiecte"), asociindu-i următorul cod:

```
on (release) {
     // arata_subiecte() va fi implementata ulterior
     _root.arata_subiecte();
}
```

În linia temporală principală (primul cadru al stratului cu numele `ActionScript`), includeți funcția care va realiza autentificarea:

```
function autentifica() {
     _root.sterge();
     _root.gol.attachMovie("OBJ_autentifica",
          "MCautentifica", 1);
     _root.gol.MCautentifica._x = 210;
     _root.gol.MCautentifica._y = 50;
}
_root.gol.setMask(masca); // mascarea din script!
```

În continuare, va trebui să construiți un *MovieClip* similar celui din imaginea de mai jos, care va avea în biblioteca simbolurilor numele de legătură `OBJ_autentifica` (după ce ați terminat de realizat acest *MovieClip*, selectați-l din bibliotecă, și din meniul contextual selectați opțiunea *Linkage*; bifați *Export for ActionScript* și *Export in First Frame*, și în câmpul identificator introduceți `OBJ_autentifica`). *MovieClip*-ul pentru autentificare va avea dimensiunile 260 x 137 pixeli și va arăta astfel:



***MovieClip*-ul folosit pentru formularul de autentificare a utilizatorului**

Textul de tip *input* din dreptul etichetei „Nume utilizator" va avea atașată variabila `utilizator`, iar cel din dreptul parolei va avea asociată variabila `parola` (în plus, la tipul liniei selectați *password*). Câmpul textual situat sub

zona de autentificare va fi de tip *dynamic* şi va avea numele de instanţă `rezultat` – în el vom afişa eventualele erori. Butonului „Autentifică" îi veţi ataşa următorul cod:

```
on (release) {
  if (this.utilizator != "") {
    loadVariables("forumengine/verificare.php", this,
      "POST");
  }
}
```

Acest script are rolul de a trimite o cerere de autentificare către programul *verificare.php* descris mai sus. Rezultatele vor fi returnate tot în acest *MovieClip* şi vor fi prelucrate în cadrul unei funcţii `onData` al cărei cod este dat şi explicat mai jos.

Ataşaţi primului cadru al *MovieClip*-ului (în linia temporală a acestuia) următorul script:

```
this.onData = function() {
  // daca utilizatorul este gasit, este returnat ID-ul
  // sau, in caz contrar, se returneaza valoarea -1
  if (this.rezultat != -1) { // rezultat diferit de -1
    // sunt memorate in _root principalele informatii:
    _root.nume_utilizator = this.utilizator;
    _root.parola_utilizator = this.parola;
    _root.ID_utilizator = this.rezultat;
    // vom permite afisarea comenzilor suplimentare
    _root.arata_meniu();
    _root.arata_subiecte();
    _root.sterge();
  } else {
    // altfel este afisat un mesaj de eroare in campul
    // situat sub obiectul de autentificare
    this.respins.text =
      "Nume utilizator sau parola incorecte !" +
       newline + "Va rugam sa incercati din nou !";
  }
};
```

Butonul de înregistrare va executa o funcţie aflată în linia temporală principală. De aceea, trebuie să îi ataşaţi următorul cod:

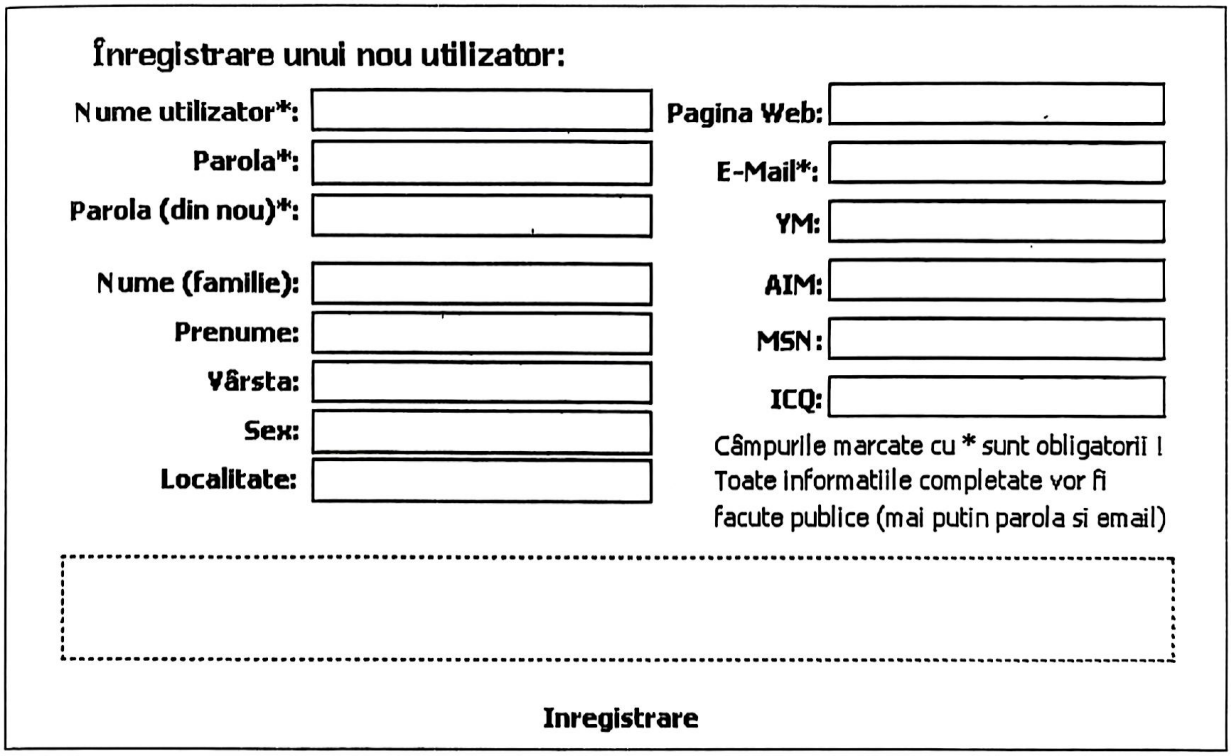
```
on (release) {
    _root.inregistrare();
}
```

Funcţia de înregistrare va aduce din bibliotecă un *MovieClip* cu identificatorul de legătură `OBJ_inregistrare` (vezi *Linkage* de mai sus) şi-l va ataşa *MovieClip*-ului `gol` (la fel cum s-a procedat şi cu `OBJ_autentifica`).

În continuare, este descrisă funcţia de înregistrare care trebuie introdusă în linia principală, după funcţia `autentifica()`:

```
function inregistrare() {
  _root.sterge();
  // atasam MovieClip-ului gol pe OBJ_inregistrare
  _root.gol.attachMovie("OBJ_inregistrare",
                        "MCinregistrare", 1);
  // apoi ii setam pozitia:
  _root.gol.MCinregistrare._x = 10;
  _root.gol.MCinregistrare._y = 10;
}
```

MovieClip-ul cu numele de legătură `OBJ_inregistrare` arată astfel:



Formularul de preluare a informaţiilor privitoare la utilizator

Câmpurile textuale sunt de tip *input*; „parola" şi „confirmare parola" vor fi de tip *password*. Fiecare *TextField* va avea câte o variabilă asociată. Ultimul câmp textual, cel situat sub cele două coloane, este de tip *dynamic* şi are numele instanţei `eroare` (a nu se confunda numele instanţei cu variabila asociată!).

Butonului „Înregistrare" trebuie să-i ataşăm un cod pentru a transfera informaţiile conţinute în formular către programul de pe server capabil să scrie în baza de date (*inregistrare.php*):

```
on (release) {
  // testam daca parolele coincid
  if (parola != parola2)
    this.eroare.text =
        "Parola nu a fost corect confirmata !";
  // testam daca a fost introdus un nume pentru
  // utilizator
  if (utilizator == "")
    this.eroare.text=
        "Nu ati introdus un nume pentru utilizator !";
  // transmitem datele catre inregistrare.php
  if ((parola == parola2) && (utilizator != "") &&
        (parola != "")) {
     loadVariables("forumengine/inregistrare.php",
        this, "POST");
  }
}
```

La fel ca în cazul autentificării, trebuie să captăm evenimentul recepţionării unor informaţii (prin intermediul funcţiei `onData`, ce va fi scrisă în primul cadru al *MovieClip*-ului):

```
this.onData = function () {
  if (rezultat != 1) {    // daca a intervenit o eroare
    this.eroare.text = mesaj_err; // afiseaza eroarea
  } else {
    _root.arata_subiecte(); // spre pagina principala
  }
};
```

În acest moment orice vizitator al forumului îşi va putea crea un cont.

3.2 Construirea obiectului de afişare a subiectelor

Prima dată când un utilizator va deschide forumul, i se vor afişa subiectele. Vom construi un *MovieClip* în care vom lista toate subiectele introduse. În stratul `ActionScript` vom continua cu introducerea următoarelor funcţii:

```
function sterge() {   // o functie de stergere a lui „gol"
  _root.gol.unloadMovieNum(1);
  _root.gol.MCmeniu.raspunde._visible = false;
}
```

```
function arata_subiecte() {
  _root.sterge();
  var XML_subiecte = new XML();// construim un obiect XML
  subiecte.ignoreWhite = true; // ignoram spatiile albe
  _root.gol.attachMovie("OBJ_subiecte", "MCsub", 1);
  aleator = Math.random()*1000000000; // numar aleator
  XML_subiecte.load
    ("forumengine/vizualizeazasubiecte.php?" + aleator);
  XML_subiecte.onLoad = function(succes) {
    _root.gol.MCsub.subiecte = XML_subiecte;
    _root.gol.MCsub.afiseaza();
  };
}
nume_utilizator = ""; // nici un utilizator
autentificat
arata_subiecte();    // afisarea subiectelor e implicita
```

În codul atașat, definim deocamdată o funcție care listează subiectele. Executarea acestei funcții va conduce la construirea unui obiect XML, setarea atributului de ignorare a spațiilor albe, eliminarea conținutului *MovieClip*-ului `gol`, după care acestuia i se va atașa un *MovieClip* existent în bibliotecă cu numele de legătură `OBJ_subiecte`. Referința către noul *MovieClip* va fi `_root.gol.MCsub`.

Efectuăm o cerere de încărcare a subiectelor prin apelul metodei `load()`, specifică obiectelor XML. Pentru că încărcarea este asincronă, va trebui să captăm evenimentul primirii tuturor datelor, lucru realizat de funcția `onLoad()` care tratează evenimentul `onLoad` specific. După ce datele au fost încărcate cu succes, XML-ul construit este transmis către *MovieClip*-ul nou atașat la scenă și este executată o comandă din interiorul său pentru a porni procesarea datelor – funcția `afiseaza()`.

Construirea *MovieClip*-ului cu numele de legătură `OBJ_subiecte` nu este chiar atât de simplă pe cât pare. Începeți prin a desena un dreptunghi de dimensiunile măștii din stratul al doilea (650 x 330 pixeli), ce va juca rol de fundal. În interiorul *MovieClip*-ului trebuie să aveți o mască de dimensiunea 638 x 319 pixeli care să mascheze un *MovieClip* vid așezat la fel ca în scena principală (în colțul din stânga sus al măștii); dați-i numele de instanță `masca`. Mascarea, precum și adăugarea unei bare de defilare vor fi realizate din `ActionScript`. În *MovieClip*-ul mascat vom atașa pe rând subiectele, și de aceea va avea numele de instanță `lista`.

Tot în componența *MovieClip*-ului (cu legătura `OBJ_subiecte`) veți mai adăuga încă un strat așezat sub cele deja construite. Ultimul strat rămas liber va fi intitulat `ActionScript` și-i vom atașa următorul cod:

```
function afiseaza() {
  // mascam din ActionScript:
  this.lista.setMask (masca);
  // construim o lista a subiectelor
  sir_subiecte = subiecte.firstChild.childNodes;
  // si un cap de tabel pentru lista subiectelor:
  this.lista.attachMovie("subiect_bar",
       "subiect_bar_", -1);
  this.lista.subiect_bar_._x = 0;
  this.lista.subiect_bar_._y = 0;
  this.lista.subiect_bar_.subiect = "Subiect";
  this.lista.subiect_bar_.autor = "Autor";
  this.lista.subiect_bar_.raspunsuri = "Rasp.";
  this.lista.subiect_bar_.data = "Data";
  this.lista.subiect_bar_.enabled = false;
  this.lista.subiect_bar_.useHandCursor = false;
  // construim lista subiectelor
  for (i = 0; i < sir_subiecte.length; i++) {
    this.lista.attachMovie("subiect_bar",
         "subiect_bar" + i, i);
    this.lista["subiect_bar" + i].id =
         sir_subiecte[i].attributes.id;
    this.lista["subiect_bar" + i].subiect =
         sir_subiecte[i].attributes.subiect;
    this.lista["subiect_bar" + i].autor =
         sir_subiecte[i].attributes.postat_de;
    this.lista["subiect_bar" + i].raspunsuri =
         sir_subiecte[i].attributes.raspunsuri;
    this.lista["subiect_bar" + i].data =
         sir_subiecte[i].attributes.data;
    // si pozitionam fiecare subiect:
    this.lista["subiect_bar" + i]._x = 0;
    this.lista["subiect_bar" + i]._y = (i + 1) * 22;
  }
  // calculam inaltimea totala pentru a verifica daca
  // este nevoie de scroll-bar
  inaltime_totala = (sir_subiecte.length + 1) * 22;
  this.attachMovie("scroll", "scroll", -2);
  this.scroll._visible = false;
  this.scroll._x = 540;
  this.scroll._y = 25;
  // scroll-bar-ul va avea o dimensiune maxima simbolizand
  // inaltimea maxima pe axa Y pe care o poate modifica:
  this.scroll.maxScroll = this.lista._height - 310;
```

```
    if (this.scroll.maxScroll > 0) {
      this.scroll._visible = true;
    }
  }
  // evenimentul onEnterFrame ne va ajuta la repozitionare:
  this.onEnterFrame = function() {
    this.lista._y = -this.valoare_scroll + 7;
  };
```

În prima parte a funcției de afișare este atașată o bară în care vom introduce titlul fiecărei coloane (subiecte/autor/răspunsuri/data). Adăugarea acestor informații este realizată prin intermediul unui *MovieClip* care va conține patru *TextField*-uri. Subiectele sunt atașate în cadrul unei structuri repetitive `for`, care va include în *MovieClip*-ul `lista` un număr de *MovieClip*-uri egal cu numărul subiectelor.

În final, se va calcula înălțimea totală și va fi atașat scenei un *MovieClip* ce va juca rolul unei bare de defilare, prin intermediul căruia vom putea deplasa lista subiectelor via captarea și tratarea evenimentului `onEnterFrame`.

Nu uitați: înainte de a trece la următorul *MovieClip*, trebuie să atașați în bibliotecă numele de legătură pentru acest obiect (`OBJ_subiecte`). După această operație, obiectul poate fi șters din scenă, el fiind adus ulterior prin intermediul lui `attachMovie()`.

Linia pe care vom atașa un subiect va avea în bibliotecă numele de legătură `subiect_bar` și va fi alcătuită dintr-un fundal și patru *TextField*-uri. Construiți pentru început un dreptunghi cu dimensiunile 520 x 20 pixeli. Convertiți dreptunghiul în *MovieClip*, efectuați un dublu *click* pentru a-l edita, apoi adăugați încă două straturi deasupra stratului existent. Dreptunghiul din scenă (cel de pe stratul al treilea) va fi convertit în *MovieClip* și-i veți da un factor de transparență de 25%. În primul strat, veți copia din nou dreptunghiul, astfel încât să se suprapună peste cel deja existent, după care îi veți seta factorul de transparență la 0 (complet transparent – acesta va juca rol de „zona *Hit*"). Tot acestui *MovieClip* (de pe primul *layer*) îi veți atașa următorul cod:

```
on (press){
  // executa functia de afisare a mesajelor din subiect
  _root.afiseaza_subiect(this._parent.id);
}
```

pentru ca atunci când este apăsat să apeleze funcția de afișare a unui subiect specific (și anume cel afișat în bara apăsată).

Mai rămâne ca pe stratul al doilea să atașați patru *TextField*-uri de tip *dynamic* pentru a afișa subiectul: primul, mai lung, va avea atașată variabila `subiect`, al doilea – `autor`, al treilea – `raspunsuri`, iar ultimul va avea

asociată variabila `data`. Puteți să reveniți în scenă și să ștergeți și acest *MovieClip*, deoarece va fi atașat din bibliotecă în timpul execuției filmului Flash. Nu uitați ca în bibliotecă să-i atașați numele de legătură `subiect_bar`.

Ultimul *MovieClip* care va intra în scenă atunci când vor fi afișate subiectele va reprezenta o bară de defilare. Pentru a construi acest element, desenați în scena principală o linie verticală cu înălțimea de 280 de pixeli. Convertiți linia în simbol și editați-o. Adăugați o bilă de 15 x 15 pixeli, care va juca rol de cursor. Convertiți bila în *MovieClip* cu punctul de înregistrare în centru și dați-i numele de instanță `bila`. Selectați bila și, după ce ați poziționat-o în capătul superior al liniei verticale, atașați-i următorul cod, care are rolul de a forța deplasarea bilei numai deasupra liniei și de a returna valoarea *scroll*-ului în *MovieClip*-ul părinte:

```
on (press) {   // la apasarea bilei se incepe „tragerea"
  startDrag(this, false, 0, 0, 0, 280);
}
on (release) { // eliberarea bilei va opri „tragerea"
  stopDrag();
}
onClipEvent (enterFrame) {
  // calculeaza & trimite pozitia in MovieClip-ul parinte
  this._parent._parent.valoare_scroll =
            this._parent.maxScroll * this._y / 280;
}
```

Obiectul alcătuit din bilă și linia verticală va avea în bibliotecă numele de legătură `scroll`.

3.3 Construirea obiectului care va afișa mesajele

Când un utilizator va dori să vizualizeze mesajele transmise în cadrul unui subiect, va apăsa bara aferentă. Atunci când este interceptat evenimentul apăsării, *MovieClip*-ul care conține subiectul va transmite funcției `afiseaza_subiect()` – aflate în `_root` – ID-ul subiectului ale cărui mesaje vor trebui afișate.

Să definim în linia principală, după `arata_subiecte()`, funcția de afișare a mesajelor:

```
function afiseaza_subiect(id) {
  _root.sterge();
  // atasam un MovieClip din biblioteca sub numele MCmesaje
  _root.gol.attachMovie("OBJ_mesaje", "MCmesaje", 1);
  // cream o structura XML a mesajelor:
  var XML_mesaje = new XML();
```

```
    XML_mesaje.ignoreWhite = true;
    aleator = Math.random()*1000000;
    // incarcam XML-ul ce contine mesajele:
    XML_mesaje.load
        ("forumengine/vizualizeazasubiect.php?IDsubiect="
        + id + "&" + aleator);
    XML_mesaje.onLoad = function(succes) {
      // trimitem mesajele in MovieClip-ul MCmesaje:
      _root.gol.MCmesaje.mesaje = XML_mesaje;
      // si invocam o functie de afisare a mesajelor:
      _root.gol.MCmesaje.afiseaza();
    };
    // activam un buton la apasarea caruia un utilizator
    // autentificat poate raspunde la subiect
    _root.gol.MCmeniu.raspunde._visible = true;
    // si memoram ID-ul subiectului pentru a fi folosit
    // in functia de adaugare a mesajului:
    _root.IDmesajCurent = id;
}
```

MovieClip-ul `OBJ_mesaje` va fi foarte asemănător cu `OBJ_subiecte`. De această dată însă, nu vom mai putea construi bare în care să arătăm mesajele (așa cum am făcut la subiecte), motivul fiind că mesajele pot avea dimensiuni diferite (în cazul subiectelor, acestea erau afișate pe un singur rând). Totuși, și de această dată ne vom folosi de *MovieClip*-ul `scroll` pentru a putea deplasa pe verticală lista subiectelor atunci când acestea sunt mai multe decât pot fi afișate în scenă.

Deschideți biblioteca și selectați simbolul cu numele de legătură `OBJ_subiecte`. Apăsați *click* dreapta și alegeți *Duplicate*. În locul numelui, introduceți noul nume al instanței (nu contează ce scrieți aici), bifați *Export for ActionScript* și *Export in First Frame*, iar în zona identificatorului specificați `OBJ_mesaje`.

Diferența va fi aceea că nu vom încărca toate barele de subiecte, de aceea ștergeți întreg ActionScript-ul pe care îl aveți definit în primul cadru (pe unul din straturile 3 sau 4 – depinde cum ați construit `OBJ_subiecte`), pentru a-l înlocui cu un altul.

Înainte de a construi noul cod ActionScript, să vedem ce impedimente vom putea întâlni: din cauza dimensiunilor diferite ale mesajelor, suprafața fondului pe care acestea sunt afișate va varia de la un mesaj la altul, chiar în cadrul aceluiași subiect (din acest motiv, suprafețele numite vor fi generate dinamic în funcție de mărimea textului). Există totuși o limită inferioară impusă de necesitatea afișării numelui celui ce a redactat un mesaj, eventual de afișarea unui avatar. Vom considera această limită inferioară de 100 de

pixeli. Trebuie să reținem de fiecare dată poziția la care s-a terminat de afișat un mesaj pentru a ști de unde începem să-l afișăm pe următorul.

Nu în ultimul rând, trebuie să ținem seama de faptul că va trebui să includem un *TextField* de tip *dynamic* care necesită și o formatare a textului pentru a fi corect afișat:

```
function afiseaza() {
  // cream o formatare pentru text
  var formatare = new
    TextFormat("Tahoma", 11, 0, false, false);
  // lista mesajelor
  sir_mesaje = mesaje.firstChild.childNodes;
  this.lista.setMask(masca);  // mascare din script
  // setam modul de trasare a background-ului
  this.lista.lineStyle(1, 0xffffff, 50);
  this.lista.beginFill(0xffffff, 40);
  // inaltimea de la care va fi afisat primul text:
  inaltime = 10;
  // construim lista mesajelor
  for (i = 0; i < sir_mesaje.length; i++) {
    // calculam inaltimea ce o va ocupa mesajul curent:
    inaltime_mesaj = formatare.getTextExtent
    (sir_mesaje[i].attributes.mesaj,
    390).textFieldHeight;
    // construim un TextField folosind valoarea
    // obtinuta:
    this.lista.createTextField("mesaj" + i,
       i + 1000, 120, inaltime, 390, inaltime_mesaj);
    // continuam cu adaugarea informatiilor,
    // pozitionarea si formatarea fiecarui TextField
    // (mesaj, nume etc):
    this.lista["mesaj" + i].text =
       sir_mesaje[i].attributes.mesaj;
    this.lista["mesaj"+i].wordWrap = true;
    this.lista["mesaj"+i].setTextFormat(formatare);
    this.lista.createTextField("autor" + i,
       i + 2000, 10, inaltime, 100, 20);
    this.lista["autor" + i].text =
       sir_mesaje[i].attributes.autor +
       "[" + sir_mesaje[i].attributes.posturi + "]";
    this.lista["autor" + i].setTextFormat(formatare);
    // adaugarea avatarului se va face intr-un
    // MovieClip cu numele de legatura „_avatar":
    this.lista.attachMovie("_avatar",
       "avatar" + i, i + 3000);
```

```
      this.lista["avatar"+i]._x = 10;
      this.lista["avatar"+i]._y = 15 + inaltime;
      this.lista["avatar"+i].continut.loadMovie
          ("avatars/" + sir_mesaje[i].attributes.avatar);
      this.lista.createTextField("data" + i, i + 4000,
          10, 83 + inaltime, 100, 20);
      this.lista["data" + i].text =
          sir_mesaje[i].attributes.data;
      this.lista["data" + i].setTextFormat(formatare);
      // calculam pozitia urmatorului mesaj:
      if (inaltime_mesaj < 100) {
        inaltime_mesaj = 100;
      }
      // trasam background-ul:
      this.lista.moveTo(0, inaltime);
      this.lista.lineTo(520, inaltime);
      this.lista.lineTo(520, inaltime + inaltime_mesaj);
      this.lista.lineTo(0, inaltime + inaltime_mesaj);
      this.lista.lineTo(0, inaltime);
      // lasam 5 pixeli intre doua mesaje consecutive:
      inaltime += inaltime_mesaj + 5;
    }
    // adaugam un scroll-bar similar celui de la subiecte
    inaltime_totala = (sir_subiecte.length + 1) * 22;
    this.attachMovie("scroll", "scroll", -2);
    this.scroll._visible = false;
    this.scroll._x = 540;
    this.scroll._y = 25;
    this.scroll.maxScroll = this.lista._height - 310;
    if (this.scroll.maxScroll > 0) {
      this.scroll._visible = true;
    }
  }
  this.onEnterFrame = function() {
    this.lista._y = -this.valoare_scroll + 7;
  };
```

Iată cum funcţionează scriptul de construire a mesajelor: în primă fază este construită o formatare ce va fi ataşată textului. Sunt setate tipul conturului şi culoarea fondului pentru un mesaj, este ascuns *MovieClip*-ul `lista` prin intermediul măştii. Apoi sunt procesate toate datele: este calculată înălţimea necesară pentru afişarea unui mesaj, este construit un câmp textual pentru a afişa corect mesajul, în alte două câmpuri sunt afişate numele utilizatorului, respectiv data postării şi este încărcat din exterior avatarul conform informaţiilor primite din PHP. Variabila `inaltime` va monitoriza

coordonata Y la care se poate adăuga un nou mesaj. În final, este atașat *MovieClip*-ul `scroll`, care ajută la derularea listei mesajelor.

MovieClip-ul în care se va încărca avatarul va avea în bibliotecă numele de legătură `_avatar`. Pentru a construi acest avatar, realizați în scenă un dreptunghi cu dimensiunile 60 x 70 pixeli. Convertiți-l în *MovieClip* și editați-l. Adăugați un nou *layer* în care copiați dreptunghiul suprapus peste cel existent. Dreptunghiul de pe stratul inferior convertiți-l în *MovieClip* și dați-i numele de `continut`. Apăsați *click* dreapta pe stratul superior și selectați *Mask*. Efectuarea acestei operații de mascare are rolul de a interzice utilizatorilor să își atașeze avatari cu o suprafață mai mare decât cea prestabilită (60 x 70 pixeli) – de fapt, avatarul se va încărca în *MovieClip*-ul `continut`. Reveniți în scena principală, de unde puteți șterge întreg *MovieClip*-ul (așa cum v-ați obișnuit, el va fi adus în scenă în timpul execuției – dar nu uitați să-i atribuiți numele de legătură `_avatar`).

3.4 Construirea meniului pentru utilizatorul autentificat

Atunci când se realizează autentificarea, în cazul în care aceasta s-a soldat cu succes, aveți apelată o funcție din `_root` cu numele `arata_meniu()`. Această funcție are rolul de a aduce în scenă un *MovieClip* conținând câteva butoane prin care un utilizator înregistrat poate interacționa cu forumul: *Subiect nou*, *Profilul meu*, *Răspunde* și chiar *Listă utilizatori* (acest ultim meniu nu va fi descris în acest capitol). Nu contează cum realizați acest meniu, important este ca el să încapă în partea dreaptă (de aceea o dimensiune potrivită ar fi 90 x 185 pixeli).

Ceea ce trebuie să mai adăugați la construirea acestui *MovieClip* este un câmp dinamic care să aibă asociată variabila `utilizator` (pentru a afișa numele utilizatorului autentificat). De asemenea, butonul *Răspunde* va avea numele de instanță `raspunde` – el nu trebuie afișat decât atunci când sunt afișate mesajele dintr-un anumit subiect și trebuie să avem o posibilitate de a-l ascunde (vezi funcția `afiseaza_subiect()` de mai sus în care se activează acest buton).

MovieClip-ul corespunzător acestui meniu va avea în bibliotecă numele de legătură `OBJ_meniu`.

Fiecărui buton îi va corespunde o serie de linii de cod ActionScript. Iată-le, pentru fiecare buton în parte:

- pentru *Subiect nou*:

```
on (release) {
   _root.adauga_subiect();
}
```

- pentru *Profilul meu*:

```
on (release) {
   _root.editare_profil();
}
```

- pentru *Răspunde*:

```
on (release) {
   _root.adauga_mesaj();
}
```

În linia temporală principală, alături de funcţiile de afişare (`arata_subiecte()`, `afiseaza_subiect()` etc.), va trebui să introduceţi funcţia de afişare a meniului:

```
function arata_meniu() {
  // daca avem un utilizator autentificat:
  if (_root.nume_utilizator != "") {
    // atasam meniul:
    _root.gol.attachMovie("OBJ_meniu", "MCmeniu", -10);
    // il repozitionam
    _root.gol.MCmeniu._x = 550;
    _root.gol.MCmeniu._y = 50;
    // adaugam numele utilizatorului in meniu
    _root.gol.MCmeniu.utilizator =
    _root.nume_utilizator;
    // si ascundem butonul de introducere a unui mesaj:
    _root.gol.MCmeniu.raspunde._visible = false;
  }
}
```

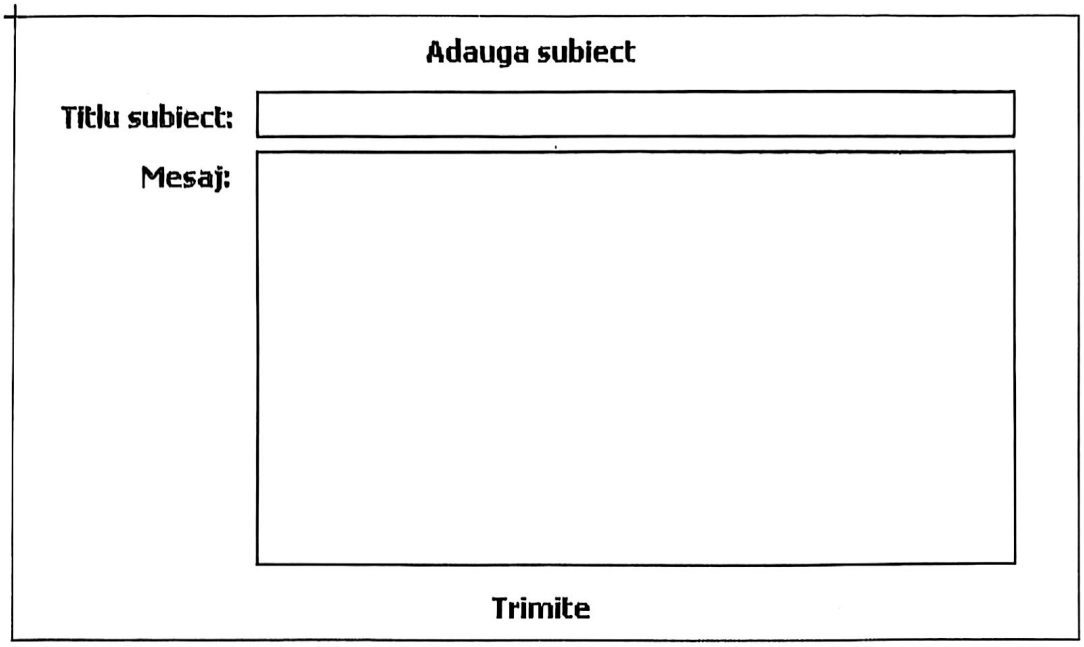
3.5 Adăugarea unui subiect

Adăugarea unui subiect este efectuată prin apelul funcţiei `adauga_subiect()`, lansată de butonul „Subiect nou" de mai sus. Funcţia va fi plasată în linia temporală principală după `arata_meniu()`. Rolul funcţiei este de a aduce în scenă un formular ce trebuie completat cu informaţiile dorite a fi transmise serverului:

```
function adauga_subiect() {
  _root.sterge();
  // aducerea din biblioteca a unui MovieClip:
  _root.gol.attachMovie("OBJ_subiect_nou",
      "MCsubiect_nou", 1);
  // vom atasa numele utilizatorului si parola pentru
  // a le trimite o data cu subiectul si primul mesaj;
  // rolul acestor informatii este de a impiedica frauda:
```

```
    _root.gol.MCsubiect_nou.utilizator =
        _root.nume_utilizator;
    _root.gol.MCsubiect_nou.parola =
        _root.parola_utilizator;
    // in final, repozitionam MovieClip-ul MCsubiect_nou:
    _root.gol.MCsubiect_nou._x = 100;
    _root.gol.MCsubiect_nou._y = 50;
}
```

Deja v-aţi obişnuit ca funcţiile ataşate liniei temporale principale să aducă din bibliotecă un *MovieClip* şi să-l ataşeze *MovieClip*-ului `gol`. Acum este preluat obiectul cu identificatorul de legătură `OBJ_subiect_nou`. Acest *MovieClip* va avea doar două *TextField*-uri şi un buton:



Formularul de editare a unui mesaj

Ambele *TextField*-uri sunt de tip *input*, al doilea având tipul liniei setat *multiline*. Variabilele ataşate celor două *TextField*-uri sunt `subiect` şi `mesaj`. Butonul „Trimite" are asociat un script care – după verificarea faptului că numele subiectului şi mesajul nu sunt vide – va transmite informaţiile necesare inserării unui nou subiect către *subiectnou.php*. Codul sursă pentru acest buton este următorul:

```
on (release) {
  if ((this.mesaj != "") && (this.subiect != "")) {
    loadVariables("forumengine/subiectnou.php",
        this, "POST");
  }
}
```

În primul cadru al *MovieClip*-ului, vom capta evenimentul sosirii confirmării de la server şi vom afişa lista subiectelor (care acum va include şi noul subiect):

```
onData = function () {
    _root.arata_subiecte();
};
```

3.6 Adăugarea unui mesaj

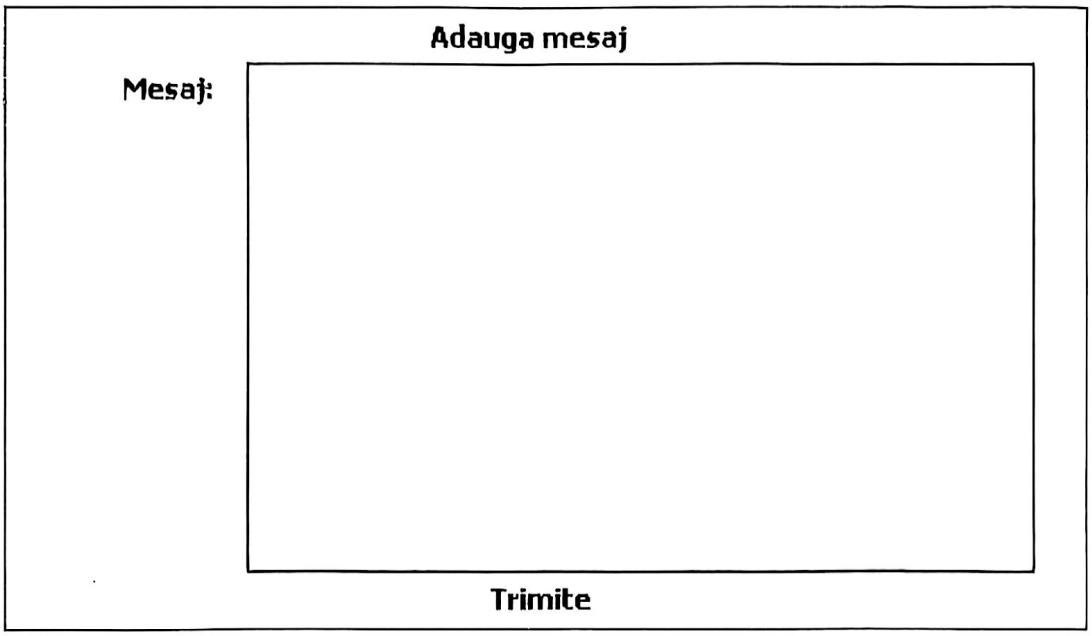
Adăugarea unui mesaj este similară cu inserarea unui subiect. În plus, pe lângă nume şi parolă, în *form*-ul în care vom scrie mesajul va trebui să adăugăm şi ID-ul subiectului (pentru a rezolva asocierea dintre mesaj şi subiect). Din acest motiv, funcţia din linia temporală principală va arăta astfel:

```
function adauga_mesaj() {
  _root.sterge();
  // atasarea MovieClip-ului din bibliteca
  _root.gol.attachMovie("OBJ_mesaj_nou",
      "MCmesaj_nou", 1);
  // adaugam numele, parola si ID-ul subiectului
  _root.gol.MCmesaj_nou.utilizator =
      _root.nume_utilizator;
  _root.gol.MCmesaj_nou.parola =
  _root.parola_utilizator;
  _root.gol.MCmesaj_nou.IDsubiect =
  _root.IDmesajCurent;
  // repozitionam MCmesaj_nou
  _root.gol.MCmesaj_nou._x = 100;
  _root.gol.MCmesaj_nou._y = 50;
}
```

Formularul pentru adăugarea unui nou mesaj, cu numele de legătură `OBJ_mesaj_nou`, va conţine doar un *input* în care va fi introdus textul răspunsului, respectiv un buton „Trimite" răspunzător pentru transmiterea mesajului şi a celorlalte informaţii către programul rulând pe server, *mesajnou.php*.

TextField-ul de tip *input* va avea asociată variabila `mesaj`, iar butonului de transmitere trebuie să îi ataşaţi următorul cod:

```
on (release) {
 // daca mesajul nu este vid
 if (this.mesaj != "") {
  // atunci va fi transmis catre mesajnou.php
  loadVariables("forumengine/mesajnou.php", this, "POST");
 }
}
```



Formularul Flash de trimitere a unui mesaj

După ce se va capta răspunsul venit de la *mesajnou.php*, se va vizualiza lista mesajelor din cadrul subiectului în care s-a făcut adăugarea. Ataşaţi primului cadru următorul script:

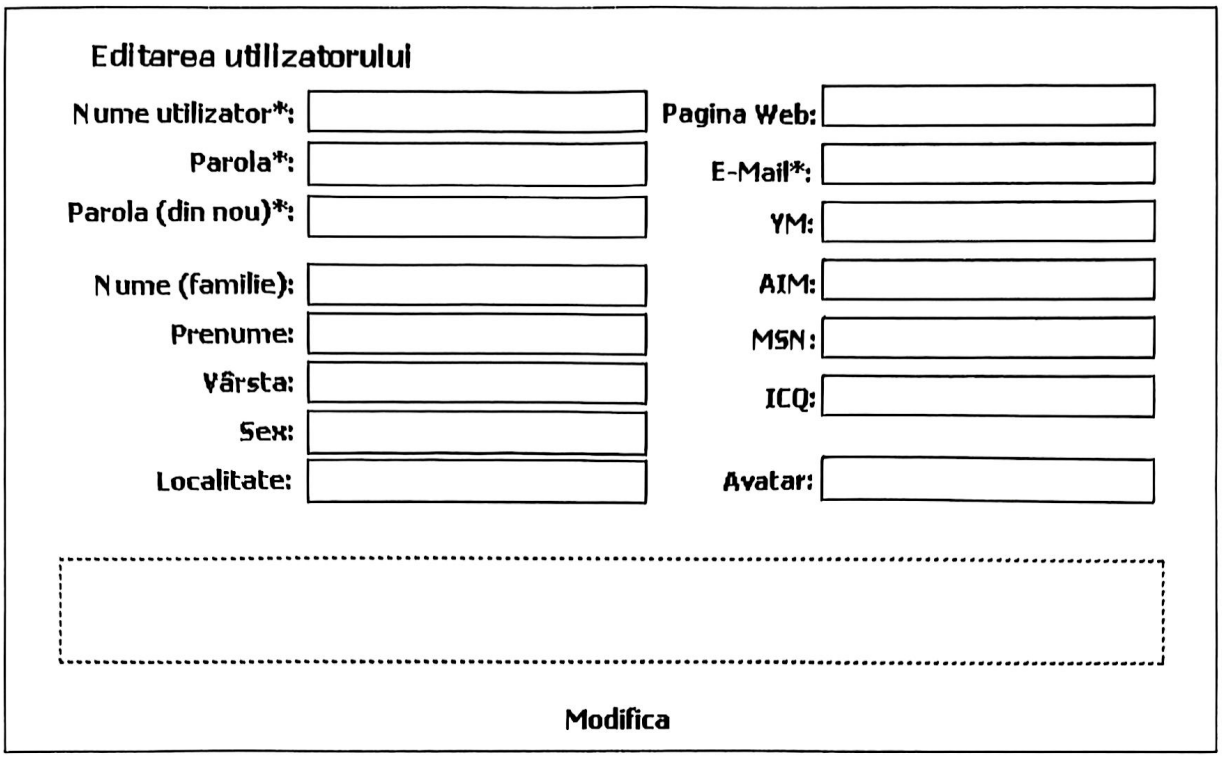
```
onData = function () {
    _root.afiseaza_subiect(this.IDsubiect);
};
```

3.7 Modificarea profilului utilizatorului

Operaţia de editare a profilului este asemănătoare cu cea de înregistrare. Totuşi, pentru a realiza această operaţie trebuie să aducem în cadrul filmului Flash toate informaţiile privitoare la utilizatorul curent. Identificatorul acestui utilizator (precum şi numele şi parola) este deja stocat în `_root` de către procedura de autentificare (sub denumirile `_root.nume_utilizator`, `_root.parola_utilizator` şi `_root.ID_utilizator`). Cunoscând ID-ul, putem afla celelalte informaţii prin apelul scriptului *vizualizeazautilizator.php*.

După ce toate informaţiile au fost recepţionate de documentul Flash, se vor ataşa unui *MovieClip* care va fi adus din bibliotecă şi care va avea numele de legătură `OBJ_editare`. Acesta va fi cu puţin diferit de `OBJ_inregistrare`, folosit pentru înregistrarea unui nou utilizator, şi din acest motiv el va fi construit prin duplicare şi apoi prin schimbarea numelui şi a identificatorului *Linkage*. Ceea ce diferă este titlul ferestrei, care acum este „Modificare utilizator" şi nu „Înregistrare", schimbarea butonului „Înregistrează" cu „Modifică" şi apariţia unui nou *TextField*: „Avatar", care va

avea ca variabilă asociată tot `avatar` (pentru inserarea unei legături către un avatar, aceste SWF-uri vor fi stocate în directorul *avatars*; în câmpul corespunzător avatarului va trebui să fie indicat doar numele fișierului).



Formularul de modificare a profilului

Altă diferență față de formularul de înregistrare este că *TextField*-ul de tip *input* din dreptul utilizatorului va fi convertit într-unul de tip *dynamic* și care nu poate fi modificat de utilizator (debifați butonul *selectable* din fereastra de proprietăți).

Codul butonului „Modifică" va transmite informațiile către *„modifica-utilizator.php"*, după ce în prealabil a testat dacă parolele coincid.

Nu uitați să interceptați datele returnate de PHP (în primul cadru al *MovieClip*-ului):

```
onData = function () {
  if (rezultat != 1) {
    this.eroare.text = mesaj_err;
  } else {
    _root.arata_subiecte();
  }
};
```

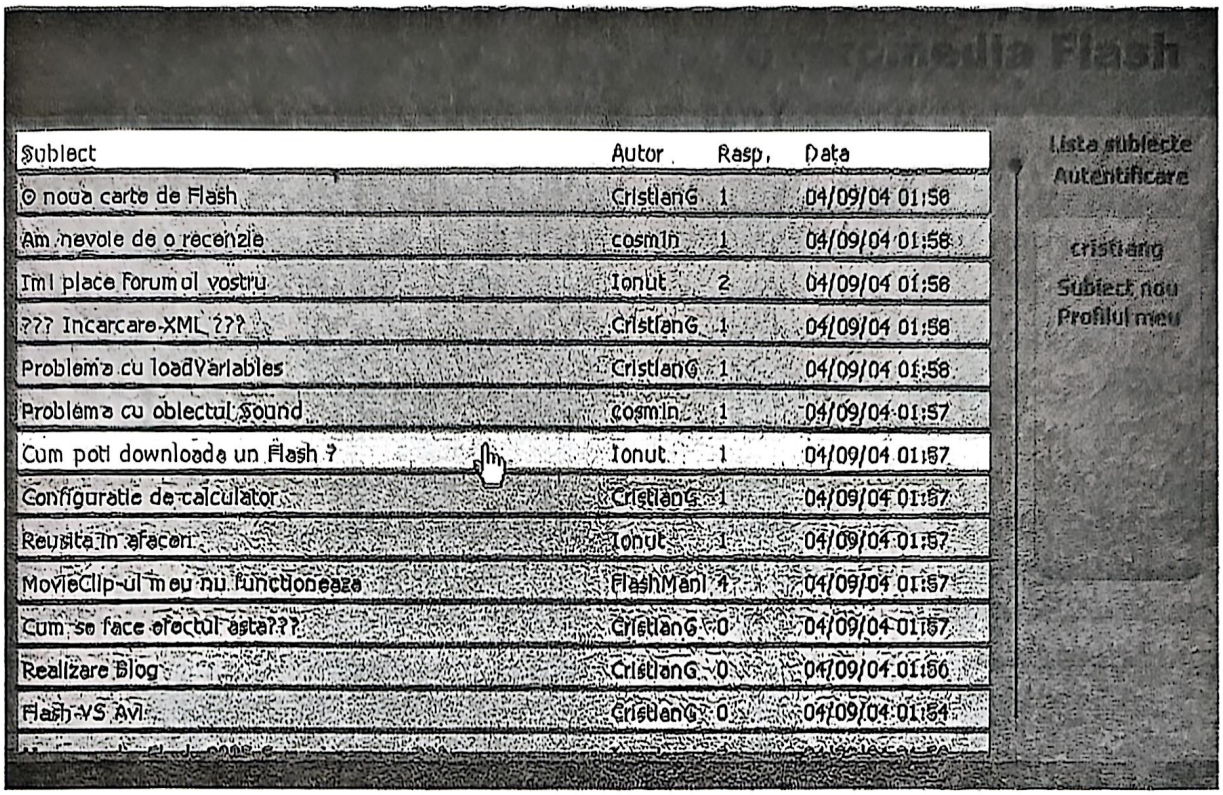
În linia temporală principală, după funcția `adauga_mesaj()`, adăugați scriptul:

```
function editare_profil() {
  _root.sterge();
  // adaugam MovieClip-ul din biblioteca
  _root.gol.attachMovie("OBJ_editare", "MCeditare", 1);
```

```
    // cream un obiect XML:
    var XML_utilizator = new XML();
    XML_utilizator.ignoreWhite = true;
    // il incarcam cu informatii despre utilizator:
    XML_utilizator.load
      ("forumengine/vizualizeazautilizator.php" +
       "?IDutilizator=" + _root.ID_utilizator);
    XML_utilizator.onLoad = function(success) {
    if (success) {
      // atasam campurilor din MovieClip-ul OBJ_editare
      // informatiile primite
      _root.gol.MCeditare.utilizator =
          XML_utilizator.firstChild.attributes.utilizator;
      _root.gol.MCeditare.nume =
          XML_utilizator.firstChild.attributes.nume;
      _root.gol.MCeditare.prenume =
          XML_utilizator.firstChild.attributes.prenume;
      _root.gol.MCeditare.ym =
          XML_utilizator.firstChild.attributes.ym;
      _root.gol.MCeditare.localitate =
          XML_utilizator.firstChild.attributes.localitate;
      _root.gol.MCeditare.aim =
          XML_utilizator.firstChild.attributes.aim;
      _root.gol.MCeditare.msn =
          XML_utilizator.firstChild.attributes.msn;
      _root.gol.MCeditare.icq =
          XML_utilizator.firstChild.attributes.icq;
      _root.gol.MCeditare.sex =
          XML_utilizator.firstChild.attributes.sex;
      _root.gol.MCeditare.varsta =
          XML_utilizator.firstChild.attributes.varsta;
      _root.gol.MCeditare.web =
          XML_utilizator.firstChild.attributes.web;
      _root.gol.MCeditare.email =
          XML_utilizator.firstChild.attributes.email;
      _root.gol.MCeditare.parola = _root.parola_utilizator;
      _root.gol.MCeditare.parola2 = _root.parola_utilizator;
      _root.gol.MCeditare.avatar =
          XML_utilizator.firstChild.attributes.avatar;
      _root.gol.MCeditare.id = _root.ID_utilizator;
      // adaugam parola curenta pentru a impiedica frauda
      _root.gol.MCeditare.parola_curenta =
          _root.parola_utilizator;
     }
    };
    _root.gol.MCeditare._x = 10;
    _root.gol.MCeditare._y = 10;
}
```

4. Concluzii

Forumul pe care tocmai l-am realizat îndeplineşte principalele funcţii ale unei astfel de aplicaţii Web. Bineînţeles, pentru a realiza un forum complet am avea nevoie de mult mai mult decât ce a fost descris aici (probabil am putea să scriem o întreagă carte pe această temă). Capitolul de faţă constituie un exemplu concludent de interoperabilitate între serverul de aplicaţii PHP şi mediul de creare a interfeţelor – utilizatorul reprezentat de Flash.



Interfaţa finală a forumului

Ne vom opri aici cu modul de realizare a forumului, dar nu înainte de a vă sugera o serie de direcţii viitoare de implementare. Aşadar, ceea ce aţi mai putea implementa în acest forum ar putea fi: un modul care să reţină numele şi parola unui utilizator şi să le salveze într-un *cookie* (acest lucru se realizează relativ uşor prin intermediul obiectului `SharedObject`), iar în plus aţi putea implementa un modul de editare a mesajelor sau de ştergere a acestora. De asemenea, ar fi util un modul de afişare a tuturor utilizatorilor, care, la selectarea unuia dintre ei, să poată furniza informaţiile pe care acesta le-a făcut publice (prin intermediul lui *vizualizeazautilizator.php*, tot ceea ce aveţi de realizat este să trimiteţi identificatorul utilizatorului în cauză).

Tot ca idei de implementări ulterioare aţi putea efectua un modul care să realizeze managementul avatarurilor (afişarea lor poate fi făcută în Flash via o listă de avataruri, iar pentru *upload*-area lor pe server puteţi folosi PHP;

mai mult, puteți recurge la un script JavaScript pentru a căuta pe discul dumneavoastră fișierul SWF local pe care intenționați să-l folosiți.

5. Comentarii bibliografice

O sursă de informații privitoare la universul Flash o reprezintă materialul cuprins în volumul C. Vârlan, *Macromedia Flash*, apărut la Editura Polirom în anul 2004. De asemenea, recomandăm vizitarea siturilor Web disponibile la adresele *http://www.actionscript.org* și *http://www.macromedia.com/devnet*.

În ceea ce privește realizarea aplicațiilor PHP, cititorii interesați pot consulta cartea S. Buraga (coord.), *Aplicații Web la cheie*, Polirom, 2003 (vezi și *http://www.infoiasi.ro/~phpapps/*), în care sunt detaliate diverse aspecte legate de conectivitatea cu bazele de date, autentificarea utilizatorilor și managementul *cookie*-urilor. Un sit care poate fi de interes este *PHP Builder – http://phpbuilder.com*.

Pentru experimentarea modului de interoperabilitate PHP-Flash, puteți să recurgeți la *Apache2Triad* – pachet incluzând versiunile Windows ale serverului Apache, ale mediilor PHP, Perl și Python, ale serverului de baze de date MySQL, plus alte instrumente folositoare programatorilor Web. *Apache2Triad* este disponibil gratuit la adresa *http://apache2triad.sourceforge.net/*.

CAPITOLUL III

Portaluri educaționale

Bogdan Manolache, toni@MVDmedia.ro

Capitolul de față începe cu o prezentare generală a portalurilor, urmată de o incursiune în lumea portalurilor educaționale. Cea mai importantă parte a capitolului este dedicată prezentării modului de implementare în PHP a unei biblioteci virtuale conținând resurse multimedia. Această bibliotecă *on-line* oferă o interfață Web de tip portal în interacțiunea cu vizitatorii.

1. La început de capitol

Informaticienii tânjesc să fie meșteri ai cuvântului sau învățați. Suntem cel puțin fascinați de limbă. Pe de o parte, reprezentăm o comunitate de profesioniști care au reușit să distileze concepte la șiruri de acronime de neînțeles. Pe de altă parte, împrumutăm limbaje bogate și descriptive de la alte discipline pentru a aduce lumină disciplinei noastre. Astfel de termeni – precum *arhitectură*, *ecologie* și *Web* – își au originea în alte domenii, dar au devenit cunoscuți sub o nouă formă în domeniul tehnologiei informației.

Unul dintre cei mai recenți termeni împrumutați este *portal*, definit în cadrul dicționarului Cambridge ca fiind „o poartă, o ușă sau o intrare; îndeosebi una care este impozantă și falnică, precum cea a unei catedrale". Ținând cont de originea ecleziastică a universității moderne, poate că este adecvat ca această ultimă metaforă a tehnologiei informației să fie una care se potrivește atât de bine cu mediul academic. Totuși, termenul *portal* reprezintă oarecum o metaforă problematică, deoarece ridică tot atât de multe probleme pe câte își propune să rezolve. Ne putem pe drept întreba: *„o poartă spre ce?"*

De la inventarea sa acum mai mult de un deceniu, Web-ul a pretins a fi o sursă de noutăți, un nou mod de a face reclamă, un manual cu reguli, un loc de formare a comunităților, o piață, o bibliotecă sau o universitate virtuală. În esență, Web-ul conține câte puțin din toate aceste lucruri și totuși nu este nici unul dintre ele. Web-ul este primul mediu din lume care permite

aproape oricui să devină un creator de conţinut, un programator, un organizator, un distribuitor, un intermediar, un cumpărător sau un vânzător. Drept urmare, Web-ul reprezintă astăzi *ultima frontieră.* Este mai puţin limitat de fizică decât de imaginaţia noastră; de aici şi tendinţa de a împrumuta metafore.

Frontierele sunt în mod firesc locuri dezordonate şi chiar nesigure, atrăgând aventurieri şi tâlhari – locuri în aşteptarea unor măsuri de organizare, chiar a unor legi. În acest context, portalul este mai mult decât o poartă. Poate că reprezintă un principiu unificator care permite organizaţiilor – şi mai ales universităţilor – să-şi mărească investiţiile în tehnică de calcul, în depozite de date, în reorganizarea procesului informaţional şi în talentul angajaţilor.

2. Portaluri

2.1 Cadrul general

Între anii 1960 şi 1970, sistemele informatice erau proiectate pentru a susţine necesităţile de procesare a informaţiilor şi a tranzacţiilor din cadrul marilor unităţi ale organizaţiilor, precum registratura, departamentele de personal şi departamentele de contabilitate. Aceste sisteme ofereau o flexibilitate limitată. Introducerea şi răspândirea, între anii 1970 şi 1980, a celei de-a patra generaţii de limbaje de programare au făcut posibilă generarea de rapoarte personalizate, destinate în special unităţilor constituind organizaţiile. În paralel, minicalculatoarele efectuau în esenţă aceleaşi sarcini pentru a face faţă necesităţilor departamentelor din cadrul instituţiilor academice.

În anii '90, dezvoltarea reţelelor, introducerea pe scară largă a Internetului şi a aplicaţiilor Web de planificare a resurselor din cadrul organizaţiilor au creat potenţialul de personalizare a modului de oferire a informaţiilor şi a serviciilor dintre unităţile centrale şi unităţile specializate din cadrul instituţiilor. Acest model a fost numit „modelul reţea". În cadrul multor instituţii, aceste inovaţii au permis o reducere semnificativă a responsabilităţilor organizaţionale, îmbunătăţiri ale nivelului serviciilor, reduceri de costuri administrative.

Internetul a fost elementul central al acestei evoluţii, în măsura în care a permis trecerea peste interfeţele „neprietenoase" moştenite de majoritatea sistemelor. O parte din vină pentru aceste moşteniri aparţine complexităţii informaţiilor instituţionale, o problemă pe care nici o tehnologie nu o poate rezolva. În acelaşi timp, Web-ul a devenit din ce în ce mai accesibil, făcând posibilă realizarea de către oricine a paginilor cu informaţii utile pe care ceilalţi să le poată accesa.

Oamenii au nevoie de informație! Dezvoltarea portalurilor este susținută de faptul că organizațiile trebuie și pot să ofere informațiile și serviciile bazate pe Web în formatul necesar utilizatorului. Aceasta este o schimbare importantă, în primul rând deoarece majoritatea organizațiilor, inclusiv universitățile, și-au organizat informațiile rareori în alt format decât cel necesar membrilor comunităților. Acest nou și interesant aspect al managementului Web oferit de portaluri se bazează pe ideea creării și administrării sistemelor informaționale al căror scop de bază este menținerea relațiilor pozitive dintre susținătorii unei instituții și acea instituție.

2.2 Ce este un portal?

Ne putem opri la definiția pe care a dat-o Howard Strauss în *Web Portals: A Home Page Doth Not a Portal Make*, Edutech Report, 2000: „un portal este evoluția fundamentală de la vechea experiență Web bazată pe entități. Portalurile reprezintă o schimbare de bază în maniera în care prezentăm informația utilizatorilor și în modul în care aceștia folosesc Internetul".

Cu alte cuvinte, un portal reprezintă coloana vertebrală a unei instituții prin care se oferă acces centralizat la toate informațiile legate de acea instituție. Portalurile devin noua fundație a aplicațiilor Web. Dezvoltarea din ultimii ani a aplicațiilor Web a condus la integrarea într-o singură aplicație a mai multor aplicații. Avantajele acestei integrări sunt prezentate în continuare:

- *Agregarea conținutului* – capacitatea de a îmbina mai multe fragmente de conținut într-o singură interfață consistentă și interoperabilă;
- *Modurile de vizualizare personalizate* – fiecare utilizator va avea acces la anumite informații, prezentate într-o formă specifică profilului de utilizator din care face parte;
- *Conținutul personalizat* – prin această facilitate, personalizarea este dusă la un nivel superior și oferă utilizatorilor posibilitatea de a stabili maniera de vizualizare a informațiilor, astfel încât să fie satisfăcute nevoile personale;
- *Modelul de securitate unificat* – utilizatorii portalului au un singur cont prin intermediul căruia vor avea acces la portal. Acest aspect asigură o autentificare unică și o securitate bazată pe roluri;
- *Facilitățile de colaborare* – în timp ce unele portaluri oferă unelte sofisticate pentru comunicare, partajarea aplicațiilor și a fișierelor, realizarea de conferințe, colaborarea se referă în principal la dezvoltarea de comunități cu interese comune, unde utilizatorii pot schimba informații bazate pe experiența lor într-un anumit domeniu de interes comun. De exemplu, un student poate cere unui profesor mai multe detalii cu privire la un proiect pe care trebuie să-l realizeze;

- *Localizarea* – se referă la personalizarea portalului (interfață și conținut) pentru regiunea în care este folosit. Această personalizare poate include tipurile de caractere folosite, limba, schimbul valutar sau formatul datei;
- *Internaționalizarea* – suportul pentru internaționalizare oferă posibilitatea dezvoltării portalului în așa fel încât să poată fi ușor localizat. În funcție de felul în care a fost proiectat portalul, acest lucru poate fi foarte ușor sau foarte dificil de realizat;
- *Ciclul de lucru* – permite utilizatorilor să parcurgă ușor un set de sarcini care provin din diverse surse de date și aplicații. De exemplu, un student poate să consulte o listă de resurse, să selecteze una sau mai multe, să le citească, să-și spună părerea despre acea resursă, chiar dacă resursa se află pe un server, iar informațiile despre resursă se găsesc în baza de date, la fel ca și opiniile altora privitoare la conținutul resurselor;
- *Auto-întreținerea* – în ultima vreme se observă implementarea în cadrul portalurilor a tehnologiilor de auto-întreținere, care se referă la administrarea și întreținerea automată a unui portal cu neimplicarea sau implicarea minimală a unui administrator;
- *Independența de platformă* – un portal ar trebui să fie compatibil nu doar cu majoritatea navigatoarelor, dar și cu diversele dispozitive (cum ar fi dispozitivele mobile) și platforme (de exemplu, alte aplicații);
- *Dezvoltarea modulară* – este un aspect specific portalurilor ca acestea să fie dezvoltate pornind de la un nucleu, fiind ulterior adăugate module pe măsură ce sunt necesare. În această manieră, dezvoltarea în ansamblu a portalului este formată din crearea de subaplicații independente, dar care comunică între ele, depanarea sau includerea de module fiind facilă.

Trebuie precizat faptul că piața de dezvoltare a portalurilor a oferit o gamă largă de facilități specifice fiecărui tip de portal. Proiectele de dezvoltare a portalurilor au devenit punctul central în cadrul majorității organizațiilor IT și nu numai.

Ca exemple de portaluri, putem menționa *Yahoo!* sau *MSN (Microsoft Network)*, care oferă – printr-o interfață unică – acces la mai multe categorii de servicii standard (poștă electronică, știri, starea vremii, cotațiile bursiere etc.). Un exemplu autohton de portal este reprezentat de *home.ro*.

În ceea ce privește soluțiile tehnologice, pot fi menționate cele disponibile în regim *open-source*, cum ar fi *eZpublish* sau *PHPNuke*, rulând sub Apache, precum și cele comerciale – de exemplu, *Microsoft SharePoint*. Universul Java oferă componente de tip portal numite *portlet*-uri.

2.3 Portaluri educaționale

Marile universități din lume punctează importanța și necesitatea unei strategii de implementare a portalurilor în cadrul instituțiilor educaționale. Portalurile educaționale sunt necesare deoarece universitățile vor fi nevoite să integreze informațiile, serviciile și infrastructura într-o interfață Web unitară și ușor navigabilă.

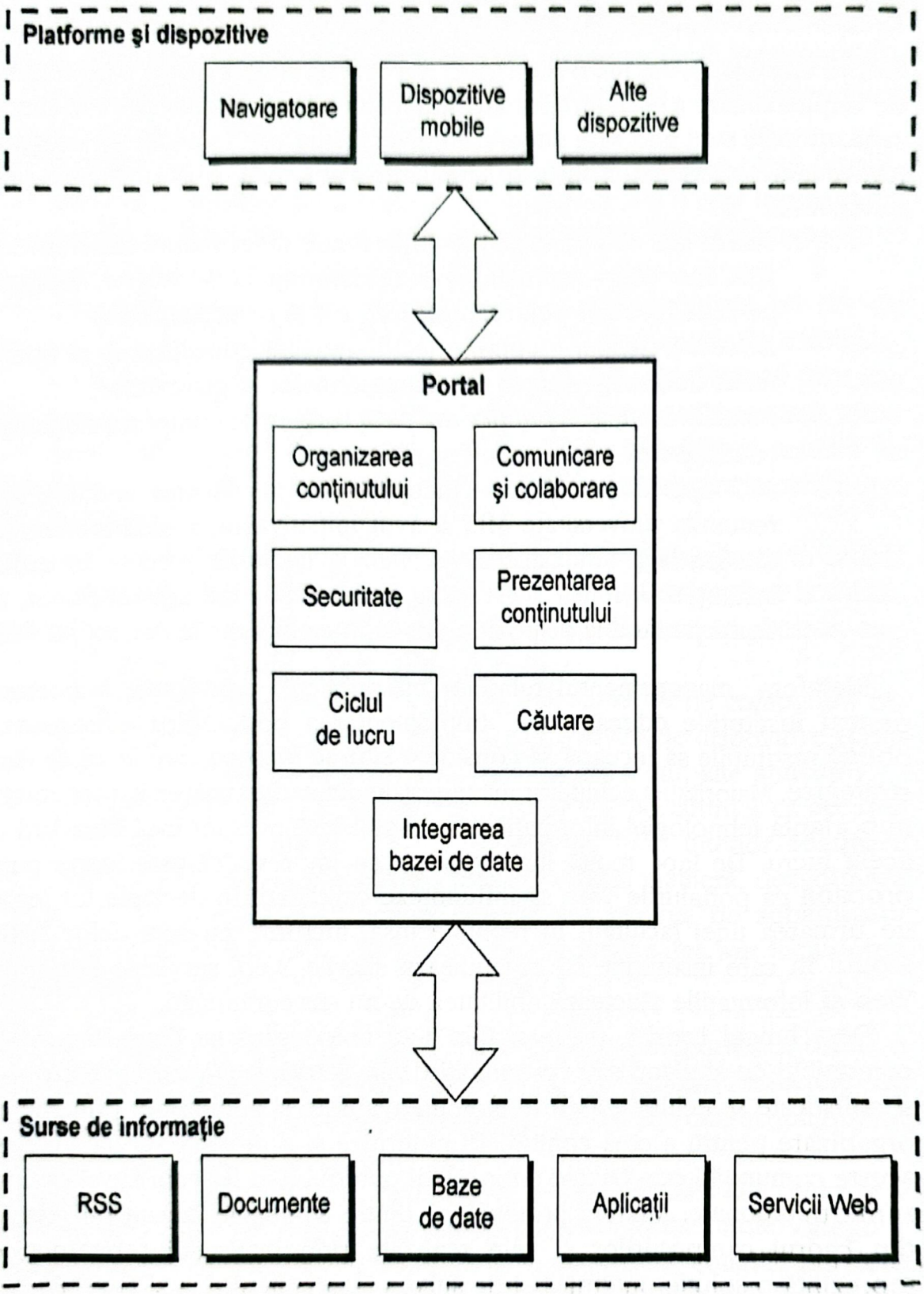
Iată în continuare câteva exemple edificatoare de portaluri educaționale:

- *BBC Learning* – reprezintă o sursă importantă de resurse, ordonate pe subiecte, atât pentru profesori, cât și pentru studenți;
- *Discovery School* – conține lecții, aplicații educaționale și numeroase informații pentru realizarea temelor și proiectelor;
- *Portalul bibliotecii publice din New York* – este o importantă resursă educațională pentru copii și părinți;
- *MIT OpenCourseWare* – începând de la sfârșitul anului 2002, renumita universitate MIT a avut inițiativa de a oferi studenților din întreaga lume acces via Web la cursurile predate în cadrul instituției. A reușit acest lucru prin intermediul acestui portal, an de an punând la dispoziție din ce în ce mai multe cursuri pe Web.

Metafora „managementul relațiilor cu publicul" este foarte importantă pentru instituțiile educaționale. Implementarea portalurilor educaționale obligă instituțiile să înceapă să considere siturile Web pe care le au ca fiind strategice. Majoritatea echipelor manageriale din cadrul universităților aprobă importanța tehnologiei informației, mulți alții însă nu sunt încă de acord cu acest lucru. De fapt, multă lume trăiește cu impresia că este foarte puțin probabil ca portalurile Web să influențeze candidații în deciziile lor legate de urmarea unei facultăți. În realitate însă, lucrurile nu stau deloc astfel. Modul în care instituțiile își organizează siturile Web, serviciile bazate pe Web și informațiile afectează abilitatea de a crea comunități.

De-a lungul istoriei, universitățile s-au autodescris ca fiind comunități: comunități de studenți sau comunități de învățământ. Organizațiile creative și sofisticate se echipează cu tehnologii avansate și adoptă noi principii de organizare pentru a crea comunități puternice și captivante. O dată create, aceste comunități pun bazele unor relații puternice, la fel cum fac și campusurile tradiționale. Mixtura armonioasă dintre strategiile bazate pe relațiile din cadrul campusurilor și cele realizate prin intermediul Internetului reprezintă o soluție puternică și de succes. Așa cum afirma și Greg Baroni: „nu este vorba despre portaluri, ci despre modernizarea educației, îmbunătățirea facilităților și asigurarea unui viitor de succes pentru universități într-un mediu total diferit"[1].

1. G. Baroni, citat într-un raport special din cadrul Simpozionului Tehnologiilor Portalurilor din anul 1999, intitulat *Into the Looking Glass: Higher Education Peeks into Portals and Sees Itself.*



Structura generală a unui portal

Două principii importante de organizare legate de portaluri sunt foarte bine subliniate de către Jenny Rickard: „Tehnologiile folosite în cadrul portalurilor oferă posibilitatea de a strânge informații provenite din mai multe surse și dau acces la aceste resurse pe baza permisiunilor pe care le are fiecare tip de utilizator"[2]. Alte aspecte importante legate de portaluri se referă la ideea accesului la informații și la servicii pe baza unei identități.

Ideea accesului pe baza identității încurajează proiectanții de portaluri să împartă virtual universitatea în funcție de rolul pe care îl are utilizatorul. Chiar dacă studenții, membrii facultății și personalul facultății pot folosi serviciile oferite de universitate, nu le utilizează în același mod. De exemplu, părinții și vizitatorii privesc instituția diferit. În aceeași manieră, arhitecții portalurilor creează diferite niveluri de acces la facilitățile oferite de portal în funcție de tipul de utilizator.

Posibilitatea de acces în funcție de drepturile pe care le are un anumit utilizator este importantă și din punctul de vedere al folosirii portalurilor pentru a îmbunătăți eficiența și productivitatea instituției. În timp ce discuțiile despre portaluri pun în special accent pe crearea de servicii pentru studenți, bazate pe tehnologiile Web, pe informații și pe comunități, o importantă instituție a adoptat soluția unui portal din motive de productivitate administrativă. Portalul reprezintă unitatea centrală a ceea ce Universitatea din California numește „arhitectura unei noi afaceri".

În esență, strategia folosirii unui portal este aceea că acesta trebuie proiectat pentru performanțe optime, acces facil, vizualizări personalizate ale informațiilor, suport și educație *on-line* extensive. În același timp, aceste sisteme permit personalului să învețe despre și să realizeze o sarcină printr-o singură tranzacție. Eliminarea tranzacțiilor intermediare dintre personalul universității și informațiile și funcționalitățile pe care acesta le dorește este cheia spre diminuarea costurilor și reducerea birocrației.

Implementarea și integrarea unui portal în cadrul unei instituții țin mai mult de schimbarea metodelor de abordare a relațiilor de administrare decât de înlocuirea echipamentelor tehnice și a softului. Pentru aceasta, universitățile trebuie să investească predominant în:

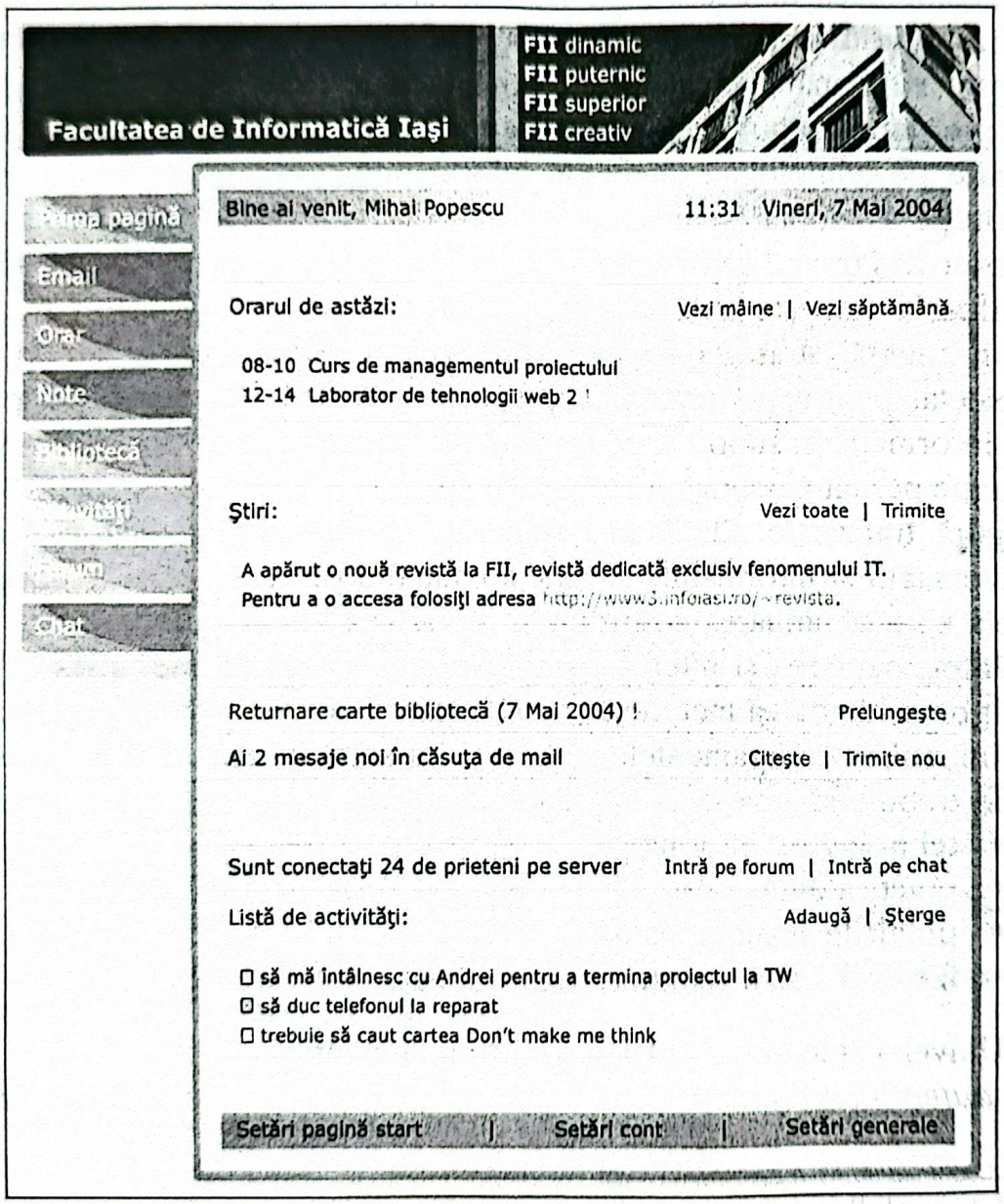
- tehnologiile informatice;
- practica afacerilor;
- politicile instituționale;
- factorul uman (cultura, organizarea).

Universitățile trebuie să reușească implementarea în portal a unei metode de *autentificare unică*. Dacă se dorește ca informațiile și serviciile oferite

2. J. Rickard, *Portals Creating Lifelong Campus Citizens*. Simpozionul Tehnologiilor Portalurilor: *Portaluri în Educația Superioară din anul 2000*.

prin intermediul portalului să fie ușor accesibile, nu se va urmări ca fiecare componentă să solicite un nume de utilizator și o parolă diferite. În același timp, portalurile trebuie să fie *stabile* și *sigure*, să identifice corespunzător utilizatorii și să le acorde doar acele drepturi pe care le au atribuite. În esență, sistemele informatice trebuie să fie capabile să determine că studentul Bogdan Manolache este chiar studentul Bogdan Manolache și are drepturi limitate, precum vizualizarea notelor obținute, iar profesorul Sabin Buraga face parte din grupul de profesori și are, de exemplu, posibilitatea de a edita notele pe care le-au obținut studenții înscriși la cursurile sale. Organizarea unei infrastructuri, a informațiilor și a serviciilor astfel încât să se comporte în acest mod este, din păcate, o sarcină dificilă.

O posibilă soluție de portal Web disponibil membrilor unei facultăți este imaginată în figura următoare.



Posibilă interfață a unui portal educațional

3. Studiu de caz: biblioteca virtuală

Cititorii care au parcurs cartea *Aplicații Web la cheie*, apărută în anul 2003 la Editura Polirom, sunt familiarizați deja cu acest proiect. Biblioteca virtuală a ajuns la o nouă versiune, având o interfață grafică îmbunătățită și oferind multe alte funcționalități.

3.1 Interfața

Aspectul grafic al acestei noi versiuni este mult îmbunătățit față de versiunea anterioară din punct de vedere estetic, dar și din punct de vedere al utilizabilității.

Din păcate, imaginea monocromă prezentată în cele ce urmează nu reflectă foarte bine înfățișarea grafică reală a sitului. Pentru o mai bună vizualizare, folosiți varianta *on-line*, disponibilă la adresa *http://www.MVDmedia.ro/elib.html*.

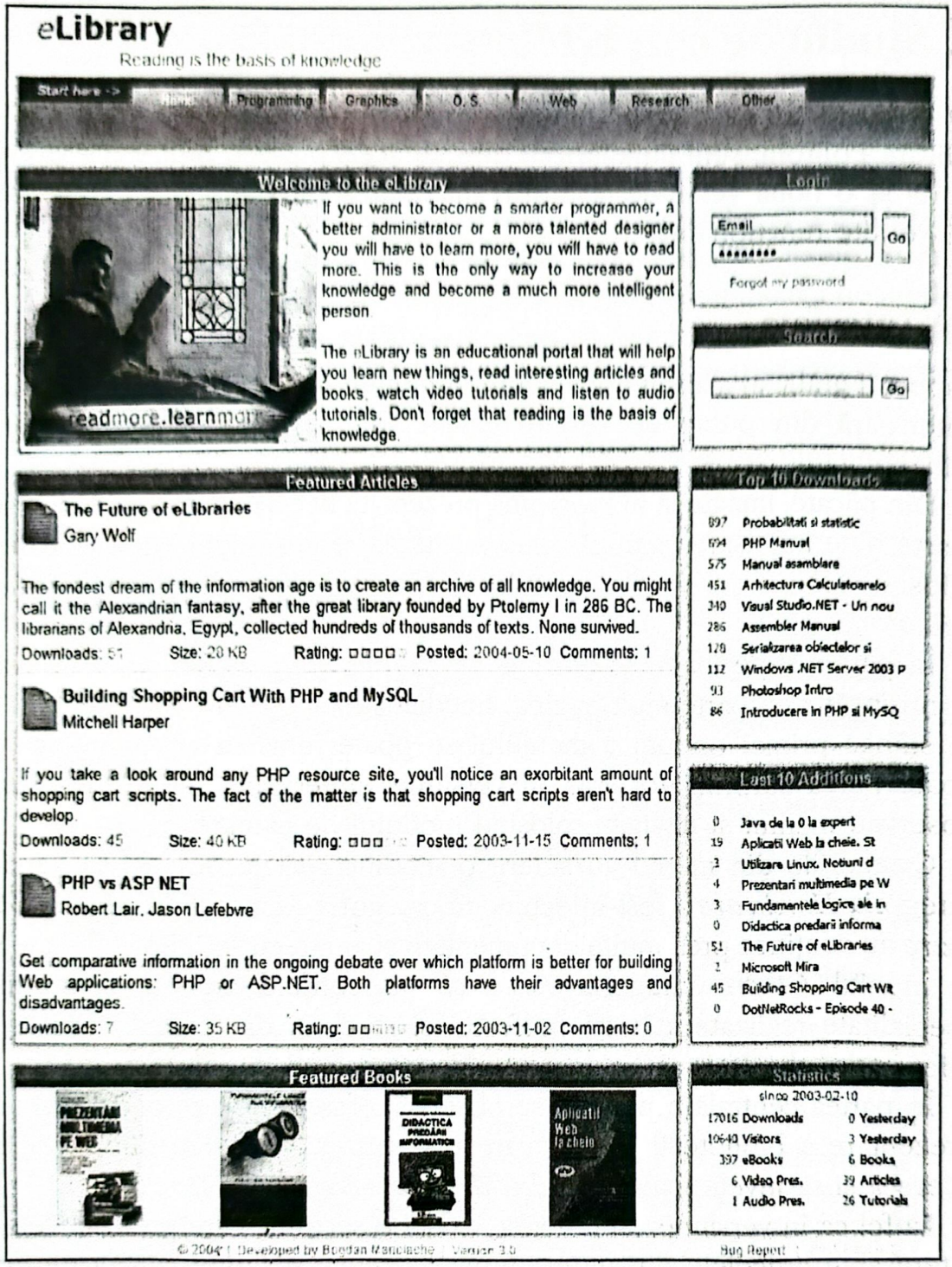
În partea superioară, se observă un *banner* simplu prin care am urmărit să nu încărcăm foarte mult pagina. Imediat sub acest *banner* se află meniul. În stânga primei opțiuni a meniului se poate remarca textul „*Start here*". Poate părea surprinzător, dar, în versiunea anterioară, mulți utilizatori nu observau meniul și navigau folosind modulul de căutare.

Categoriile din meniu au suferit o modificare față de vechea versiune: categoria *Hardware* a fost înlocuită cu categoria *Research*. Categoria *Hardware* nu conținea prea multe documentații și am trecut-o drept sub-categorie la meniul *Other*. Categoria *Research* va conține articole și studii de specialitate realizate atât de profesori și studenți din cadrul Facultății de Informatică din Iași, cât și de persoane provenind din alte facultăți.

În partea centrală a paginii, se observă o fereastră care conține o scurtă prezentare a bibliotecii și prin care încercăm să sugerăm utilizatorului să citească și să învețe mai mult („*read more. learn more*").

La fel ca în versiunea anterioară, există o secțiune care prezintă cele mai interesante articolele din ultima lună. Se remarcă faptul că fiecare articol conține, pe lângă titlu și autor, o scurtă introducere, numărul de accesări, mărimea, punctajul pe care l-a primit de la cititori (vom vorbi despre acest aspect mai târziu), data la care a fost publicat, precum și numărul de comentarii pe care le-a primit.

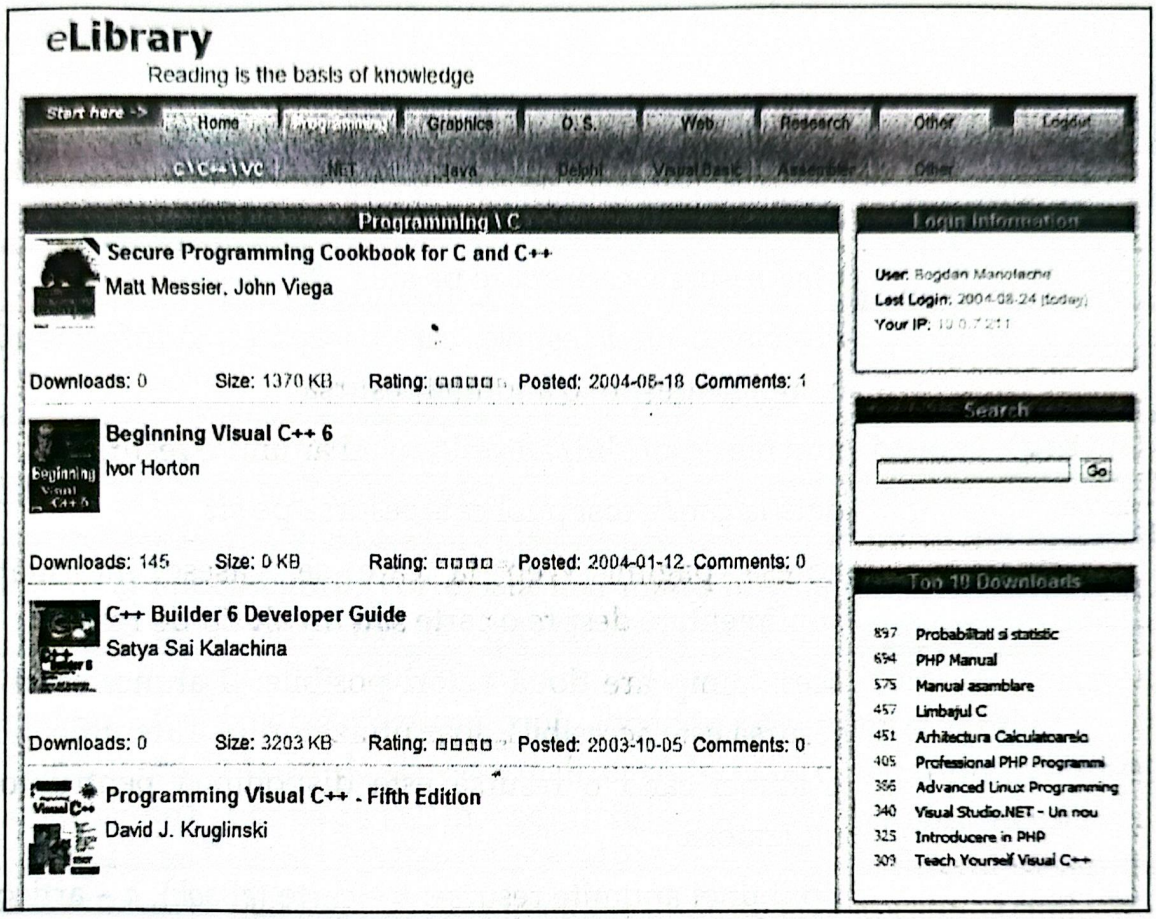
O noutate importantă pentru biblioteca virtuală este posibilitatea de a fi prezentate și cărți publicate de profesorii din cadrul facultății. Pe prima pagină sunt prezentate ultimele patru astfel de cărți. La selectarea unei cărți vor apărea informațiile complete despre carte, precum editura la care a fost publicată, adresa Web și o scurtă descriere.



Prima pagină a bibliotecii virtuale

În partea din dreapta a paginii se găsesc: modulul de autentificare, modulul de căutare, lista celor mai accesate resurse, lista ultimelor 10 resurse adăugate și statisticile legate de sit. Tot ca o noutate, un utilizator are posibilitatea de a raporta, prin intermediul opțiunii *Bug Report*, o problemă pe care o are în exploatarea sitului, reducându-se astfel timpul de rezolvare. De asemenea, vizitatorul poate solicita o anumită resursă de care are nevoie și care nu se găsește încă în bibliotecă.

Atunci când un utilizator ajunge pe sit, are drepturi limitate față de un utilizator autentificat. De exemplu, nu poate da o notă (calificativ) la o carte, nu poate scrie un comentariu despre un articol pe care l-a citit și nu are acces decât la un număr restrâns de resurse. După autentificare, utilizatorul are acces la toate resursele și funcționalitățile pe care le oferă situl. Se observă că în locul modulului de autentificare au apărut datele despre utilizatorul autentificat, iar în meniu a apărut o nouă opțiune, pentru deconectare.



Pagina cu resurse după autentificare

Pe pagina cu resurse, în coloana din partea dreaptă, am mai adăugat o secțiune care listează ultimele 10 resurse accesate de către utilizatorii care se află pe sit. În acest mod, un utilizator intenționând să parcurgă un document, dar încă nehotărât asupra tematicii, se poate inspira din alegerile făcute de ceilalți vizitatori. Aceluiași scop îi servesc și secțiunile cu cele mai accesate resurse și cu ultimele 10 resurse adăugate.

Alături de cele două tipuri de utilizatori prezentate până acum (utilizator normal și utilizator autentificat) mai există un tip de utilizator care se ocupă, firesc, de administrarea sitului: *administratorul*. După autentificare, administratorul poate adăuga, edita sau șterge resurse sau poate studia statisticile legate de sit (de la ce adrese au intrat ultimii utilizatori, ce resurse au accesat, de câte ori au accesat situl etc.).

3.2 Stocarea informațiilor. Baza de date

Pentru realizarea bibliotecii a fost necesară o singură bază de date MySQL. În continuare, vom prezenta structura bazei de date și fragmentul de cod care realizează conectarea.

Cea mai importantă tabelă din baza de date este cea care păstrează informațiile despre resurse; aceasta se numește *books* și are următoarea structură:

Denumire	Descriere
`bookid`	număr unic care identifică o anumită resursă (ID) – cheie primară
`title`	titlul resursei care apare pe sit
`description`	descrierea unei resurse, care va apărea la informațiile complete despre o anumită resursă
`author`	numele autorului/autorilor unei anumite resurse
`date`	data la care a fost publicată resursa pe sit
`link`	adresa paginii Web la care se găsesc informații suplimentare despre o carte sau un alt tip de resursă
`public`	acest câmp are două valori posibile: 0 atunci când o resursă este accesibilă doar utilizatorilor autentificați și 1 atunci când o resursă este disponibilă pentru toți utilizatorii
`type`	tipul unei anumite resurse: *b* – carte (*eBook*), *a* – articol, *t* – tutorial, *v* – prezentare video, *au* – prezentare audio, *book* – carte fizică (scrisă de profesori)
`publishedby`	editura care a publicat o anumită carte
`lastdownload`	data și ora ultimei accesări pentru o resursă

Structura tabelei *books*

Fiecare resursă face parte din una sau mai multe categorii și sub-categorii. Iată în continuare structura tabelei care conține categoriile și a tabelei care realizează legătura între resurse și categorii:

Denumire	Descriere
`catid`	număr unic care identifică o anumită categorie (ID)
`name`	titlul categoriei, care apare în meniu

Structura tabelei *categories*

Denumire	Descriere
`bookid`	număr identificând resursa în tabela *books*
`cat`	număr specificând categoria în tabela *categories*
`subcat`	număr care identifică subcategoria în tabela *subcat*

Structura tabelei *bookcategories*

Tabela *bookcategories* realizează legătura între o resursă (prin ID-ul acesteia) și o anumită categorie (via ID-ul acesteia) și subcategorie (tot prin intermediul ID-ului acesteia). Pentru resursele care fac parte din mai multe categorii și subcategorii, vor exista mai multe înregistrări în această tabelă.

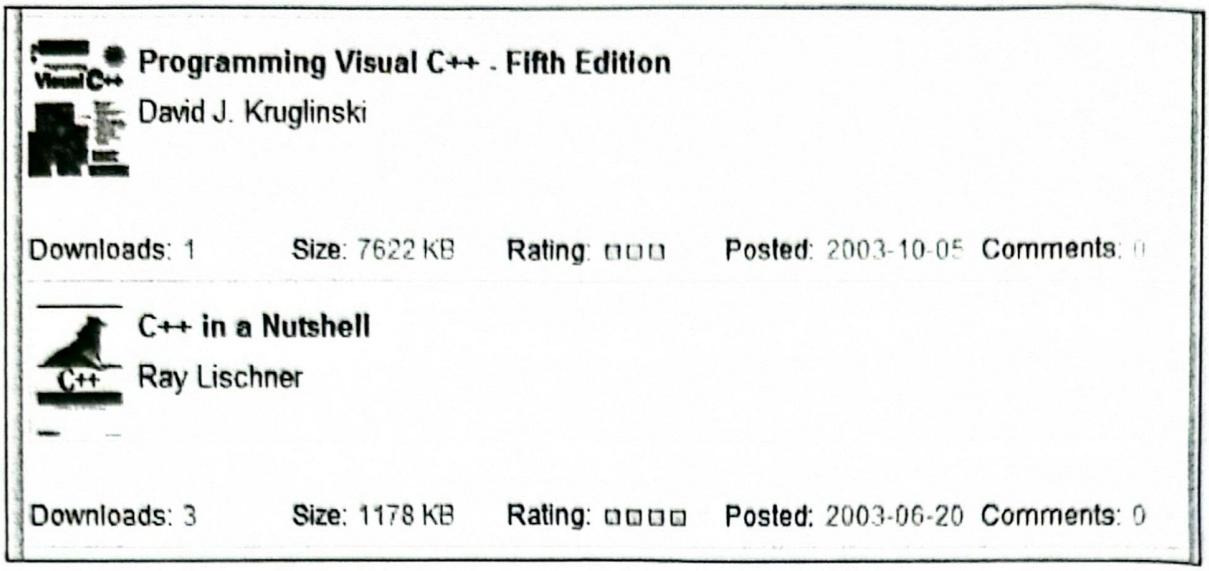
Acestea sunt cele mai importante tabele necesare pentru realizarea bibliotecii virtuale. Iată, în continuare, restul tabelelor folosite: *comments*, *download*, *featuredart*, *featuredbook*, *ipaction*, *ips*, *users* și *visitors*.

Conectarea la baza de date se realizează în mod clasic, prin intermediul funcției `mysql_connect()`, în cadrul fișierului *connect.php*. Acesta va fi inclus în toate fișierele bibliotecii virtuale. Este indicat ca fișierul să fie localizat cu un nivel mai sus față de directorul unde se află stocate paginile Web, pentru ca – în cazul în care interpretorul PHP nu funcționează – utilizatorii să nu aibă acces la informațiile de autentificare din baza de date. În acest mod prevenim alterarea neautorizată a datelor. Detalii pot fi găsite în volumul S. Buraga (coord.), *Aplicații Web la cheie*, Editura Polirom, Iași, 2003.

3.3 Listarea resurselor

Cea mai importantă funcționalitate a aplicației este afișarea resurselor cu toate informațiile despre acestea: titlu, autor(i), descriere, număr de accesări, mărime, notă, data la care a fost publicată, număr de comentarii.

Codul care face posibilă selectarea informațiilor din baza de date este prezentat în cele ce urmează. Se verifică lista opțiunilor care au fost selectate de utilizator, iar în funcție de aceste opțiuni este formată o frază SELECT care



Listarea resurselor

alege din baza de date doar cărţile dorite. În primul caz, se verifică dacă s-a ales căutarea unei cărţi pe baza titlului, apoi dacă s-a selectat căutarea tuturor cărţilor unui autor. Dacă nici una dintre opţiunile precedente nu este aleasă, atunci se listează implicit toate cărţile din categoria şi subcategoria selectate.

```
if(isset($search) && $search != "")  //cautare simpla
{
  $searchWords = separate_words($search, ' ');
  $i = 0;
  while( $i < sizeof($searchWords) )
  {
    $searchFraze .= "%" . $searchWords[$i] . "%";
    $i++;
  }
  $where = " WHERE title LIKE '$searchFraze' AND
    bookcategories.bookid=books.bookid ";
  $orderBy = " ORDER BY date DESC ";
}
else if(isset($asearch) && $asearch != "")
{ //cautare autor
  $where = " WHERE author LIKE '%$asearch%' AND
    bookcategories.bookid=books.bookid ";
  $orderBy = " ORDER BY date DESC ";

}
else
{
  $where = " WHERE cat=$c AND subcat=$s AND
```

```
    bookcategories.bookid=books.bookid ";
  $orderBy = " ORDER BY date DESC ";
}?>
```

După ce am realizat selectarea cărților care vor fi afișate, trebuie să listăm informațiile specifice pentru fiecare carte în parte. Pentru aceasta, trebuie să interogăm baza de date folosind comanda SELECT pe care tocmai am format-o. Vom prelua din baza de date informații despre o anumită carte precum titlul, autorul, descrierea, data la care a fost adăugată sau numărul de comentarii pe care le-a primit. Generăm apoi tabelul care va afișa acele informații pe pagină. Codul care realizează aceste acțiuni este următorul:

```
$fArtQuery = mysql_query("SELECT * FROM books,
  bookcategories ".$where.$orderBy." limit $n,10");
while( $fArt = mysql_fetch_array($fArtQuery ))
{
  $showBar++;
  $faDescription = $fArt["description"];
  $faTitle = $fArt["title"];
  $faAuthor = $fArt["author"];
  $faId = $fArt["bookid"];
  $faDate = $fArt["date"];
  $faType = $fArt["type"];
  $comments = 0;
  $commQuery = mysql_query("SELECT * FROM comments
    WHERE bookid=$faId");
  while( $comm = mysql_fetch_array($commQuery ))
    $comments++;
?>
<table border="0" width="100%" id="table73">
  <tr><td width="9%" rowspan="3">
    <img border="0" src="
      <?php
        if(file_exists('bookImages/' . $faId .
          '.gif'))
          echo 'bookImages/' . $faId . '.gif';
        else
          if(file_exists('bookImages/' .
            $faId.'.jpg'))
            echo 'bookImages/' . $faId . '.jpg';
      ?>
  " width="46" height="60"></td><td colspan="2">
  <font size="2" color="#084B87"><b>
  <?php if($faType=="book") { ?>
    <a href="" title="View more details about the book"
```

```
        onClick="window.open('bookdetails.php?id=
        <?php echo $faId;
        ?>', 'newWindow', 'toolbar=no, status=no,
              menubar=no, scrollbars=yes, titlebar=no,
              resizable=true, width=640,height=600');
              return false">
        <?php } else { ?>
      <a href='<?php echo
        "programming.php?asearch=$asearch&" .
        "search=$search&c=$c&s=$s&n=$n&itemId=$faId"; ?>'
        title="Download">
    <?php } ?>
    <?php echo $faTitle; ?></a></font></b>
    </td></tr><tr><td><font size="2" color="#084B87">
    <?php
      $authors = separate_words($faAuthor, ',');
      $i = 0;
      while( $i < sizeof($authors) )
      {
        if($i != 0)
          echo ", ";
        echo "<a
          href='programming.php?asearch=$authors[$i]'
          title='Search for other writings by this
          author'>$authors[$i]</a>";
        $i++;
      }
    ?> </font>

  </td>
```

Să analizăm pe scurt liniile de cod prezentate mai sus. În bucla `while` parcurgem rezultatele care au fost găsite în baza de date, conform selecției dorite. Aceasta înseamnă că, pentru fiecare resursă care se potrivește criteriilor (de exemplu, resursele care fac parte dintr-o anumită categorie), acest cod va fi parcurs o dată. Extragem din tabela *books* informațiile specifice pentru fiecare resursă în parte, apoi din tabela *comments* aflăm comentariile făcute pentru o resursă, iar în cele din urmă generăm codul HTML care va fi afișat pe pagină.

Dacă vom migra codul la noua versiune PHP 5, avem posibilitatea să recurgem la funcționalitățile oferite de noua interfață de programare `mysqli` („*MySQL Improved*") și la iteratori – o altă noutate folositoare. Detalii sunt disponibile la adresa *http://www.php.net/mysqli*.

Atunci când am prezentat structura tabelei *books*, care conține cele mai importante informații despre resursele din bibliotecă, s-a putut observa că

există o coloană cu titlul *author*. Corespunzător fiecărei resurse, această coloană conține o înregistrare de genul *„Sabin Buraga"* sau *„Sabin Buraga, Bogdan Manolache"*, pentru resurse care au doi (ori mai mulți) autori. La afișarea autorilor unei resurse, imediat sub titlu, există posibilitatea ca la selectarea unui autor să se listeze toate resursele ale cărui autor este. Pentru aceasta, trebuie să separăm autorii cu ajutorul funcției `separate_words()` prezentate în continuare:

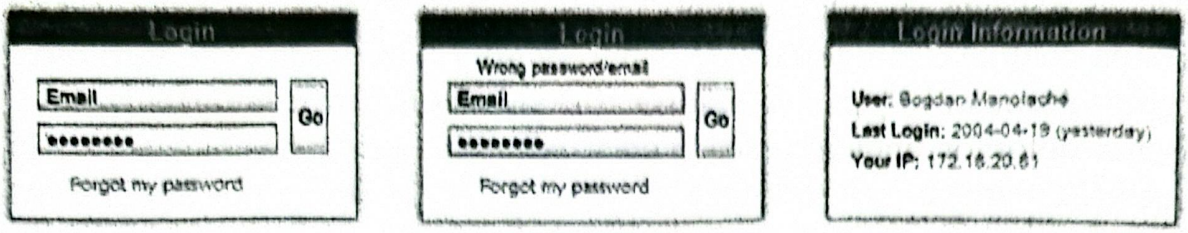
```
function separate_words($words, $separator)
{
 $i = 0;
 if ( strpos($words, $separator) )
 {
  while( strpos($words, $separator) )
  {
   $pos = strpos($words, $separator);
   $word[$i] = trim(substr($words, 0, $pos));
   $i++;
   $words = substr_replace($words, "", 0, $pos + 1);
  }
 }
 $word[$i] = trim($words);
 return ($word);
}
```

Această funcție nu mai era necesară dacă ar fi existat o tabelă *authors* care să conțină toți autorii, fiecare cu un număr unic de identificare, la fel ca în cazul categoriilor. Însă, deoarece în baza de date din vechea versiune existau peste 500 de înregistrări care conțineau autorii în acest format, am preferat realizarea unei funcții. Aceasta primește ca parametri șirul care conține autorii și separatorul (în cazul nostru, virgula), returnând un tablou cu atâtea elemente câți autori sunt în șirul primit ca parametru de intrare.

3.4 Modulul de autentificare

Aplicația permite autentificarea utilizatorilor, oferind astfel facilități suplimentare. După autentificare, un utilizator va avea acces la toate resursele din cadrul bibliotecii, inclusiv la cele cu acces privat, la care un utilizator neautentificat nu are acces, poate adăuga un comentariu la o carte pe care a citit-o sau poate evalua o carte cu o notă de la 1 la 5. Pentru modulul de autentificare vom folosi variabile de tip sesiune, care trebuie inițializate.

Atunci când un utilizator introduce informațiile de autentificare (adresa de *e-mail* și parola) este rulat scriptul din fișierul *login.php*, care verifică dacă utilizatorul cu aceste informații de autentificare există în baza de date.



Ipostazele ferestrei de autentificare

În cazul în care au fost introduse date greşite, este reîncărcată pagina de la care s-a încercat autentificarea şi se afişează un mesaj de eroare.

```
if($_POST['email'] == "" || $_POST['passwd'] == "")
{
  header("Location: " . $page . "?&pass=Empty");
  exit;
}
include("functions.php");

//verificam daca exista informatii de autentificare
$sql = "select * from users where email = '" .
  $_POST['email'] . "' and password = '" .
  $_POST['passwd'] . "' ";

$result = mysql_query($sql);
//redirectam prin Location, in caz de eroare
if(mysql_num_rows($result) != 1)
{
  header("Location: " . $page . "?&pass=Wrong");
  exit;
}

//pastram in variabile de sesiune informatiile necesare
$_SESSION['sesid'] = session_id();
$_SESSION['mail'] = $_POST['email'];
$_SESSION['password'] = $_POST['passwd'];
$res = mysql_fetch_array($result);
$_SESSION['name'] = $res["name"];
$_SESSION['usertype'] = $res["type"];
$_SESSION['lastLogin'] = $res["lastLogin"];

$em = $_POST['email'];
$today = date("Y-m-d");
mysql_query("UPDATE users SET lastLogin='$today'
  WHERE email='$em'");
//redirectam prin Location trimis in antetul HTTP
header("Location: " . $page);
```

În secțiunea de cod prezentată mai sus (fișierul *login.php*), după ce am verificat existența unui utilizator ale cărui date de autentificare se potrivesc cu cele introduse, păstrăm în variabila de sesiune informațiile referitoare la acel utilizator (numele, adresa de *e-mail*, tipul de utilizator, ultima dată la care s-a autentificat). După autentificare, utilizatorul are drepturi suplimentare față de un utilizator normal.

3.5 Managementul voturilor

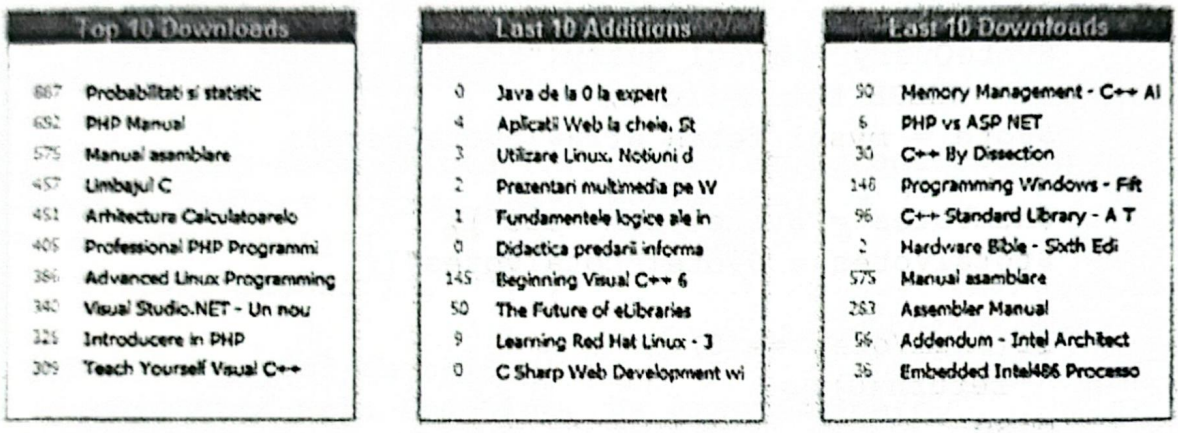
Pentru a contoriza și vizualiza voturile acordate de utilizatori, vom folosi funcțiile al căror cod sursă este furnizat mai jos:

```
//returneaza nota unei anumite carti
function getVoteNumber($id)
{
     $voteQuery = mysql_query("SELECT * FROM download
          WHERE bokid=$id");
     $vote = mysql_fetch_array($voteQuery);

     $numVotes = $vote["numvotes"];
     $totalVotes = $vote["totalvotes"];

     if($numVotes == 0)
       return(0);
     else
     {
        if( ( round($totalVotes/($numVotes))) == 5 )
          return(5);
        if( ( round($totalVotes/($numVotes))) == 4 )
          return(4);
       ...
        if( ( round($totalVotes/($numVotes))) == 1 )
          return(1);
      }
}

//insereaza in baza de date nota acordata unei carti
function insertVote ($id, $vote)
{
     mysql_query("UPDATE download SET
        numvotes=numvotes+1,
        totalvotes=totalvotes+$vote WHERE bokid=$id");

}
```

3.6 Topul celor mai accesate resurse

Dacă un vizitator venit pe sit doreşte să-şi lărgească orizontul cunoştinţelor din domeniul informaticii, dar nu ştie titlul unei anumite cărţi, atunci poate alege una din topul celor mai descărcate resurse, fiind asigurat de calitatea conţinutului acelei resurse tocmai datorită numărului mare de descărcări.

Pentru realizarea acestui top, trebuie să selectăm resursele în ordine descrescătoare după numărul de descărcări şi să afişăm doar primele 10 dintre acestea, lista astfel obţinută reprezentând topul celor mai descărcate 10 resurse din biblioteca virtuală. Pentru fiecare resursă, vom afişa numărul de descărcări şi doar primele 26 caractere din titlu, pentru a încăpea în celula alocată. Fiecare titlu de resursă reprezintă şi o legătură pentru descărcare, simplificând în acest fel sarcina utilizatorului, care nu mai trebuie să caute acea resursă.



Listele celor mai accesate cărţi, a ultimelor resurse adăugate şi a ultimelor accesate

Codul PHP care generează topul este prezentat în continuare. Se formează fraza SELECT de căutare a resurselor sortate descrescător după numărul de accesări.

```
<table border="0" width="100%" id="table35"
  cellspacing="0" cellpadding="0">
  <tr>
     <td width="4"> </td>
     <td width="34"> </td>
     <td width="3"></td>
     <td> </td>
  </tr>
  <?php
     $topQuery = mysql_query("SELECT * FROM books,
        download WHERE bokid=bookid AND type!='book'
        ORDER BY accesed DESC LIMIT 0, 10");
     while( $top = mysql_fetch_array($topQuery ))
```

```
        {
          $topAccesed = $top["accesed"];
          $topTitle = $top["title"];
          $topId = $top["bookid"];
    ?>
    <tr>
        <td width="4"> </td>
        <td width="34" align="center">
            <font face="Tahoma" size="1" color="#00CF36">
            <?php echo $topAccesed; ?></font>
        </td>
        <td width="3"></td>
        <td>
            <a title="<?php echo $topTitle;?>"
               href='<?php echo "programming.php" .
                 "?asearch=$asearch&search=$search" .
                 "&c=$c&s=$s&n=$n&itemId=$topId"; ?>' >
            <font face="Tahoma" size="1">
              <?php echo substr($topTitle, 0, 26); ?>
            </font>
            </a>
        </td>
    </tr>
    <?php
        } // while
    ?>
    <tr>
      <td colspan="4">
        <span style="font-size: 12pt"> </span>
      </td>
    </tr>
  </table>
```

Pentru fiecare înregistrare, se memorează numărul de accesări, titlul și numărul unic de identificare a resursei, pe care îl vom trimite fragmentului de cod utilizat pentru realizarea acțiunii de *download* a unei resurse selectate. Iată în continuare liniile de cod sursă:

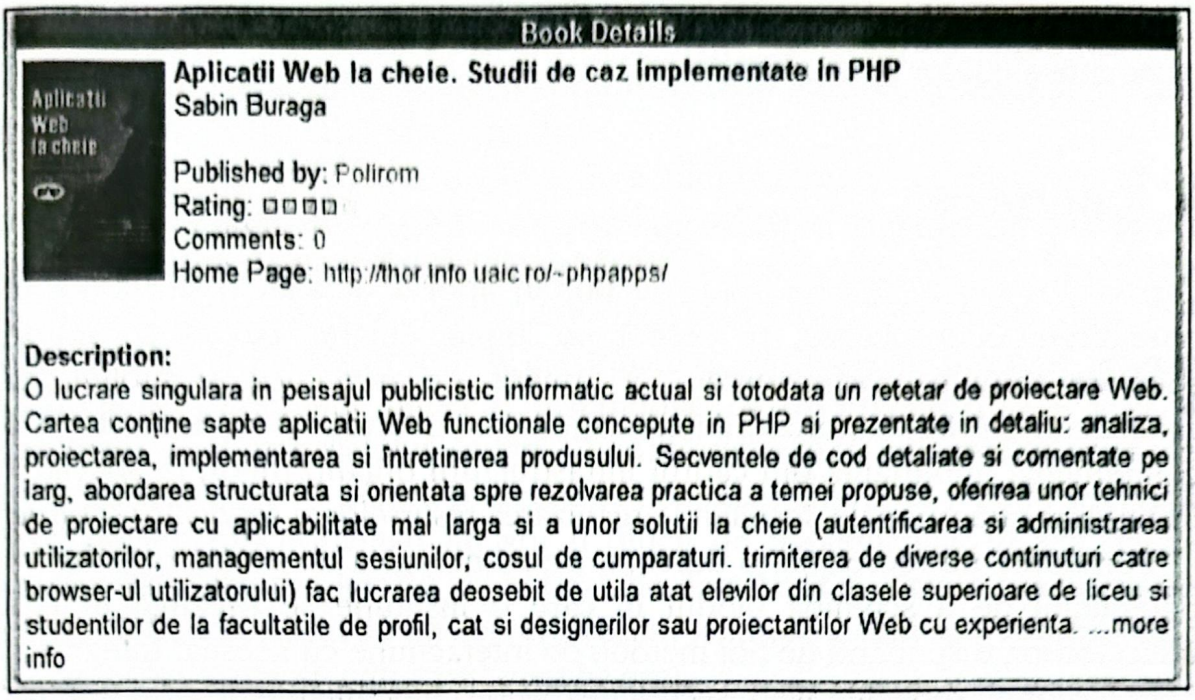
```
if( idExists($itemId) )
{
    $itemCat = categoryName($itemId);
    $itemSubcat = subcategoryName($itemId);
    $iLink = itemLink($itemId);
    $iTitle = itemTitle($itemId);
```

```
    if( $iLink == "" )
       $itemLink = $path .
          "bookdir/$itemCat/$itemSubcat/$iTitle.zip";
    else
       $itemLink = $path .
          "bookdir/$itemCat/$itemSubcat/$iLink";

    if(file_exists($itemLink))
    {
     //incrementam nr. de accesari pt. aceasta resursa
     mysql_query("UPDATE download SET accesed=accesed+1
        WHERE bokid=$itemId");

     $lastDownloadTime =
     strtotime(date("Y-m-d H:m:s"));
     mysql_query("UPDATE books
        SET lastdownload=$lastDownloadTime
        WHERE bookid=$itemId");

     //inregistram data si ora accesarii
     $ipdate = date("F d");
     $iptime = date("H:i:s");

      mysql_query("insert into ipaction
          set ip='$REMOTE_ADDR', allowedip=1,
          ipdate='$ipdate', iptime='$iptime',
          ipbook='$iTitle', bookid=$itemId ");

?>
<script language="javascript">
    window.open("<?php echo $itemLink; ?>")
</script>
<?php
    }//if
}
```

3.7 Descrierea resurselor

În cazul anumitor cărți este disponibilă și descrierea succintă a conținutului acestora, pentru o mai bună informare asupra subiectului pe care îl abordează.

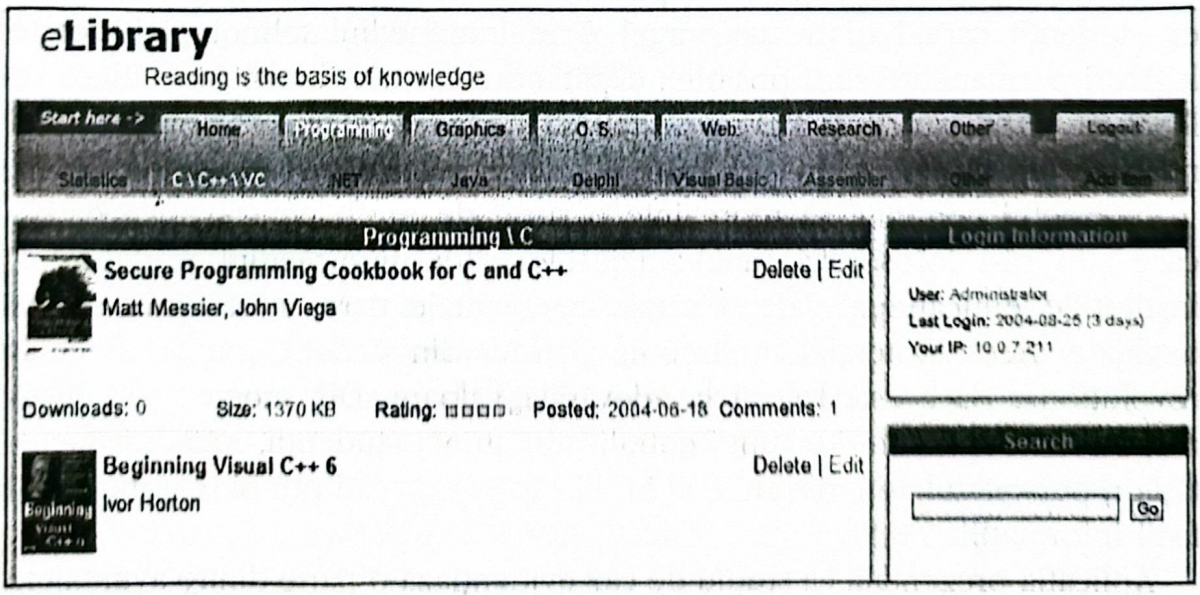
De asemenea, este prezentă și o legătură spre situl Web dedicat acelei cărți în vederea obținerii de informații suplimentare, precum prețul sau modalitatea de achiziționare.



Informații suplimentare pentru o anumită carte

3.8 Modulul de administrare

Pentru administrarea oricărei aplicații Web este nevoie de un modul de administrare oferind instrumentele necesare asigurării funcționării în condiții optime a sitului. În cazul de față, rolul administratorului este cel de a adăuga, edita sau șterge resurse. În acest mod, biblioteca este actualizată în permanență cu un conținut proaspăt. De asemenea, modulul de administrare oferă și statistici referitoare la accesarea resurselor.



Interfața modulului de administrare

Modulul de administrare este foarte asemănător cu restul paginilor. Se folosesc aceleași linii de cod, oferindu-se însă și opțiunile de editare și

ștergere a resurselor. Recomandăm cititorilor să conceapă acest modul pe baza cunoștințelor acumulate.

4. Concluzii

Capitolul a prezentat conceptul de portal, apoi a detaliat particularitățile portalurilor educaționale, cu accent pe implementarea bazată pe soluții software gratuite (*open-source*). S-a urmărit, de asemenea, și sublinierea avantajelor oferite de portalurile educaționale. Portalurile s-au dovedit a fi o componentă importantă din cadrul infrastructurii organizațiilor, în special a instituțiilor educaționale, facilitând organizarea resurselor și serviciilor *on-line*.

Portalurile nu reprezintă o descoperire tehnologică de ultimă oră, dar au potențialul de a schimba modul în care o instituție se prezintă în fața utilizatorilor, dispunând de noi metode de interacțiune cu aceștia. Integrarea portalurilor cu serviciile Web, securitatea Web și modalitățile de administrare a resurselor reprezintă o soluție completă și robustă.

În ciuda investițiilor inițiale necesare implementării unui portal, o dată implementat, acesta oferă servicii integrate care elimină intervenția personalului administrativ prin automatizarea sarcinilor și interoperabilitatea dintre module. Rezultă un dublu avantaj, de a garanta serviciile oferite utilizatorilor oricând aceștia au nevoie (24 de ore pe zi, 7 zile pe săptămână) și de a economisi resursele umane necesare activităților de rutină.

Prin intermediul portalurilor, universitățile reușesc păstrarea unei legături permanente cu utilizatorii interesați de procesul educațional și chiar cu absolvenții. Orice instituție educațională ar trebui să continue comunicarea cu studenții care i-au trecut pragul. Prin intermediul tehnologiei, aceste legături permanente sunt posibile, căpătând chiar noi dimensiuni. Ideea ca legătura dintre studenți și universități să fie una „pe viață" poate deveni realitate datorită portalurilor.

Internetul continuă să reprezinte o sursă de profit pentru organizațiile care știu să-l folosească în avantajul lor. Prin intermediul portalurilor, instituțiile educaționale beneficiază de avantaje care nu trebuie să fie neglijate. Acest potențial nu trebuie pierdut din vedere, putând fi chiar revoluționar dacă este folosit la adevărata valoare. Din moment ce informațiile din surse diferite sunt centralizate, atunci studenții, membrii facultăților, personalul administrativ, alți utilizatori interesați pot obține mult mai facil informațiile de care au nevoie.

Aplicația prezentată ca studiu de caz evidențiază o parte dintre avantajele pe care le oferă portalurile educaționale. Via o interfață de tip portal, utilizatorii au posibilitatea de a accesa diverse resurse multimedia (în special cărți). Mai mult decât atât, vizitatorii pot acorda voturi ori pot vizualiza

diverse detalii privitoare la documentele disponibile. Modulul de administrare facilitează managementul resurselor stocate, oferind și diverse statistici privitoare la accesul pe sit.

Pe viitor, se urmărește o dezvoltare permanentă a acestui proiect cu intenția de a oferi studenților o resursă importantă de informare. Din punct de vedere tehnologic, informațiile privitoare la unele cărți ar putea fi preluate direct de la *Amazon.com*, grație serviciilor Web puse la dispoziție via SOAP (*Simple Object Access Protocol*). Informațiile privitoare la ultimele resurse disponibile pe sit vor putea fi disponibile ca documente RSS – vezi și capitolul VII al materialului de față. Datorită suportului îmbunătățit oferit de noua versiune PHP 5 pentru prelucrarea documentelor XML și pentru servicii Web, aceste funcționalități nu vor fi dificil de implementat. O altă îmbunătățire ar putea fi migrarea de la baza de date MySQL către o bază de date nativă XML, folosindu-se un server precum *Xindice*, *eXists* sau *Tamino XML Server* (a se vizita și *http://xml.apache.org*).

5. Comentarii bibliografice

La sfârșit de capitol, cititorul interesat de aspectele legate de portaluri sau de implementarea acestora folosind serverul de aplicații PHP poate parcurge și următoarele resurse tipărite:

1. B. Breedlove, *Web Programming Unleashed*, Sams, 2003.
2. B. Bulger, J. Greenspan, *MySQL/PHP Database Applications*, Wiley & Sons, 2004.
3. S. Buraga, *Tehnologii Web*, Matrix Rom, București, 2001.
4. S. Buraga, *Proiectarea siturilor Web*, Polirom, Iași, 2002.
5. S. Buraga (coord.), *Aplicații Web la cheie*, Polirom, Iași, 2003.
6. W. O. Galitz, *The Essential Guide to User Interface Design*, Wiley & Sons, 2002.
7. D. Goodman, *JavaScript Bible*, Hungry Minds, 2003.
8. R. Katz, *Web Portals and Higher Education*, Educause, 2003.
9. J. C. Meloni, *PHP Essentials*, Premier Press, 2003.
10. W. Richardson, D. Avondolio, *Professional Portal Development*, Wiley & Sons, 2004.
11. A. Trachtenberg, *Upgrading to PHP 5*, O'Reilly & Associates, 2004.
12. L. Welling, *PHP and MySQL Web Development*, Sams, 2003.

CAPITOLUL IV

Modelarea și reprezentarea relațiilor dintre tabele

Sorin Bahmătă, sib@infoiasi.ro

Aplicația *DataR* reprezintă un sit Web dezvoltat în limbajul PHP ce oferă posibilitatea de a modela și reprezenta metadatele asociate structurii bazelor de date relaționale (mai exact, a bazelor de date MySQL), aici incluzând și detalii despre relațiile între tabele. Modelarea relațiilor se realizează via funcțiile PHP, iar reprezentarea acestora – prin intermediul limbajului *Resource Description Framework* (RDF).

1. Preambul

Pe parcursul acestui capitol vom detalia modul în care a fost gândită și implementată aplicația *DataR*. Vom începe prin a descrie conceptele privitoare la relațiile stabilite între tabele, urmând a arunca o privire asupra limbajul RDF standard de reprezentare a *metadatelor* (date despre date). Ultima parte analizează aplicația și propune soluții pentru implementare, aici fiind descrise structura fișierelor RDF generate de aplicație (denumite în continuare *RelDF*), modelarea relațiilor, precum și modul de interacțiune cu aplicația *RelVis* – prezentată în cadrul capitolului următor –, care oferă o vizualizare grafică a fișierelor *RelDF* prin generarea de documente *SVG* (*Scalable Vector Graphics*).

2. Relațiile dintre tabele

2.1 Introducere

Ca programatori, una dintre alegerile cele mai importante, chiar și atunci când dezvoltăm aplicații simple, este modelul corect de stocare a datelor, adică felul în care aplicația va stoca (și accesa) datele. Alegerea unui anumit model depinde de necesitățile aplicației, iar în cazul aplicațiilor cu o cantitate semnificativă de date este necesară folosirea bazelor de date, ceea ce asigură

accesarea şi manipularea informaţiilor într-o manieră rapidă şi eficientă. Acest lucru depinde însă şi de *integritatea datelor*, asupra căreia ne vom opri în continuare, având drept studiu de caz serverul de baze de date *MySQL*.

De ce MySQL? Un prim răspuns ar fi că MySQL este disponibil gratis. Dar acest aspect nu este singurul important, MySQL oferind multe alte avantaje:

- *Software open-source*: pachetul MySQL include şi codul sursă;
- *Suport SQL*: după cum sugerează şi denumirea, MySQL suportă *SQL* (*Structured Query Language*), limbaj standard de nivel înalt folosit pentru a crea interogări (*queries*) pentru datele din tabelele unei baze de date;
- *Performanţă şi siguranţă*: MySQL este rapid şi sigur, chiar şi într-un context de utilizare de mari dimensiuni şi complex;
- *Uşor de folosit*: MySQL este relativ simplu. De fapt, bazele de date MySQL nu sunt decât un grup de directoare în cadrul cărora tabelele bazei de date sunt păstrate în fişiere separate;
- *Portabil*: MySQL e disponibil atât pe platforma Windows, cât şi pe platforma UNIX (Linux).

2.2 Integritate referenţială

Pentru a explica noţiunea de *integritate referenţială*, vom urma paşii creării tabelelor şi a relaţiilor între acestea. Înainte de a trece la crearea de tabele, trebuie explicat ce este un index. Un *index* reprezintă o listă separată, sortată după un număr de câmpuri (minim 1) din cadrul unei tabele. Pentru a evidenţia efectul indecşilor asupra bazelor de date, să considerăm o tabelă fără indecşi. O astfel de tabelă este în esenţă similară unui fişier text, întrucât motorul bazei de date o parcurge secvenţial. Înregistrările nu sunt introduse într-o anumită ordine, ci arbitrar, caz în care – dacă dorim să căutăm o anumită informaţie – motorul va efectua o scanare completă a tabelei, activitate lentă şi ineficientă, mai ales dacă sunt doar câteva potriviri cu ceea ce dorim să căutăm.

Acum să considerăm o tabelă indexată (care are cel puţin un index definit). În loc să parcurgă întreaga tabelă, motorul scanează indexul pentru a căuta potriviri cu ceea ce dorim. Cum indexul este o listă sortată, această scanare este efectuată mult mai rapid. Indexul ghidează motorul către înregistrările potrivite şi, drept urmare, nu este necesară o traversare completă a tabelei.

Firesc, vine întrebarea: de ce nu sortăm pur şi simplu tabela? Acest lucru este eficient dacă dorim să căutăm informaţii bazate doar pe un anumit câmp, însă în practică, situaţia aceasta este destul de rară. Din moment ce o tabelă nu poate fi sortată după mai multe câmpuri în acelaşi timp, cea mai bună soluţie este de a crea un index separat de tabelă. Însă reala utilitate a indecşilor îşi spune cuvântul atunci când căutăm prin mai multe tabele simultan. Dacă tabelele nu ar fi indexate, atunci motorul ar căuta informaţiile prin toate combinaţiile posibile făcute între înregistrările tabelelor (două tabele cu 500 de înregistrări fiecare oferă 250.000 de combinaţii posibile!).

Indexarea scade însă semnificativ timpul de căutare. Motorul va căuta în indexul primei tabele pentru a găsi potrivirea cu ceea ce se doreşte a se obţine, apoi continuă cu indexul următoarei tabele şi aşa mai departe. Cu alte cuvinte, motorul furnizează direct înregistrările relevante.

O *cheie primară* este un index special. Ea este folosită pentru a identifica în mod unic înregistrările şi pentru a relaţiona tabelele între ele, oferindu-ne modelul relaţional al bazei de date. Fiecare tabelă poate să aibă numai o cheie primară. Totodată, o cheie primară nu poate fi nulă.

O *cheie unică* reprezintă, aşa cum spune şi numele, un grup de câmpuri ale căror valori sunt unice. Cheile primare sunt unice prin definiţie. Diferenţa dintre cele două tipuri de chei este că o tabelă poate avea mai multe chei unice, iar cheile unice pot avea şi valori nule.

Pentru a putea fi create tabele, este necesar să vă conectaţi la un server MySQL cu un *nume de utilizator* şi o *parolă*, iar o dată autentificaţi, să aveţi acces la o bază de date care le va conţine. Implicit, la instalare, MySQL creează două baze de date: *mysql* şi *test*, plus utilizatorul *root*, care are drepturi de administrare a serverului. Baza de date *mysql* este esenţială pentru serverul MySQL. În cadrul ei sunt incluse tabele cu informaţii privitoare la celelalte baze de date, dar şi la utilizatori şi drepturile lor. Baza de date *test* este folosită în special de utilizatorii anonimi (care se conectează la server fără nume şi parolă) şi iniţial nu conţine nici o tabelă. În continuare, vom considera această bază de date în cadrul căreia vom crea trei tabele aflate în relaţie, *studenti*, *cursuri* şi *note*, după cum urmează:

```
mysql> use test;
Database changed
mysql> CREATE TABLE studenti
    -> (
    -> id INT(11) NOT NULL,
    -> nume VARCHAR(20),
    -> prenume VARCHAR(20),
    -> PRIMARY KEY (id)
    -> );
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE cursuri
    -> (
    -> id INT(11) NOT NULL,
    -> nume VARCHAR(30),
    -> credite INT(2),
    -> PRIMARY KEY (id)
    -> );
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE note
    -> (
    -> cursID INT(11),
    -> studentID INT(11),
    -> nota INT(2),
    -> INDEX (cursID),
    -> INDEX (studentID)
    -> );
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Crearea celor trei tabele

În cadrul tabelelor *studenti* şi *cursuri*, am definit, folosind clauza PRIMARY KEY, drept cheie primară câmpul `id`, care reprezintă un număr întreg identificând în mod unic fiecare student, respectiv curs. Tabela *note* are doi indecşi, definiţi pe câmpurile `cursID`, respectiv `studentID`, folosind sintaxa INDEX (sau analogul KEY). Cu ajutorul acestor indecşi vom defini relaţii către celelalte două tabele, şi anume: `cursID` va reprezenta identificatorul (ID-ul) unui curs, iar *studentID* – ID-ul unui student. Să introducem câteva înregistrări în cele trei tabele:

```
mysql> INSERT INTO studenti VALUES
    -> (1,'Bahmata','Sorin'),
    -> (3,'Bulancea','Catalin');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> INSERT INTO cursuri VALUES
    -> (1,'Tehnologii WEB',6),
    -> (2,'Programare JAVA',6),
    -> (4,'Baze de date',5);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> INSERT INTO note VALUES
    -> (2,1,10),
    -> (1,3,10),
    -> (1,1,9),
    -> (4,3,10);
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql>
```

Inserarea datelor în tabele

Pe baza acestor date, putem deduce că „Bahmătă Sorin" a luat notele 10 şi 9 la obiectele „Programare Java", respectiv „Tehnologii WEB", iar „Bulancea Cătălin" – nota 10 la obiectele „Tehnologii WEB" şi „Baze de date", dar nu numai: putem afla numărul de credite ale fiecărui student, notele studenţilor de la un anumit obiect etc.

Să presupunem că se schimbă valorile din cadrul câmpului `id` din tabela *studenti*, şi anume din „1" în „111", iar în tabela *cursuri*, din „1" în „123". În acest moment nu mai putem spune că ar exista o relaţie între tabelele *note-studenti*, *note-cursuri* pe baza înregistrărilor, deoarece `studID`-ul „1" nu se găseşte în lista valorilor câmpului `id` din cadrul tabelei *studenti*, iar `cursID`-ul „1" nu se află în lista valorilor câmpului `id` din cadrul tabelei *cursuri*. Totodată, avem de-a face cu înregistrări invalide în cadrul tabelei *note*, conform relaţiilor dintre tabele.

Prima şi cea mai directă dintre soluţii este ca atunci când se schimbă `id`-ul din cadrul tabelei *studenti* sau *cursuri*, acest lucru să se realizeze şi în cazul tabelei *note*. Evident, pot apărea probleme dacă modificarea nu este efectuată corect (schimbăm `id`-ul din tabela *studenti* din „1" în „111", dar în

tabela *note* modificăm `studentID` din „1" în „11"). O situaţie asemănătoare se întâlneşte şi în cazul în care o anumită înregistrare din tabela *studenti* sau *cursuri* este ştearsă: trebuie parcursă tabela *note* pentru a elimina şi aici înregistrările corespunzătoare. Acest lucru este simplu în cazul de faţă, deoarece avem doar o singură tabelă de actualizat. Însă în practică, mai ales în cazul aplicaţiilor complexe, situaţia ar conduce deseori la modificarea înregistrărilor a câtorva zeci de tabele. Din fericire, există o soluţie mai eficientă: *integritatea referenţială*.

Pe scurt, *integritatea referenţială* înseamnă că, atunci când o înregistrare dintr-o anumită tabelă referă o înregistrare corespondentă dintr-o altă tabelă, aceea înregistrare corespondentă va exista. Integritatea referenţială se bazează pe noţiunea de *chei străine*. O cheie străină este un index dintr-o anumită tabelă care referă un alt index (de obicei, o cheie primară) din altă tabelă. Relaţiile determinate de cheile străine sunt descrise drept relaţii de tip *părinte-copil*. În cazul exemplului nostru, *studenti* şi *cursuri* sunt tabele părinte, iar *note* este tabelă copil.

În MySQL, pentru ca un câmp (sau mai multe) să fie cheie străină, trebuie folosită sintaxa FOREIGN KEY. Acest lucru se poate face la crearea tabelei sau după ce a fost creată. Putem defini chei străine pentru orice tip de tabele MySQL, incluzând şi tipul *MyISAM*, tip cu care sunt create tabelele dacă nu precizăm altfel, dar, în cazul acestor tipuri, ele nu realizează absolut nimic. Cheile străine sunt folosite doar pentru tabele de tip *InnoDB*, primul tip de tabele utilizat pentru a defini chei străine, în scopul de a asigura integritatea datelor (pentru mai multe detalii despre tipurile de tabele MySQL, vizitaţi *http://www.mysql.com*). Pentru a crea o cheie străină, trebuie îndeplinite următoarele condiţii:

- ambele tabele trebuie să fie de tip *InnoDB*;
- trebuie folosită sintaxa:

```
FOREIGN KEY (numeCampCheieStraina) REFERENCES numeTabela
(numeCampCheiPrimara)
[ON DELETE { CASCADE | NO ACTION | SET DEFAULT | SET NULL }]
[ON UPDATE { CASCADE | NO ACTION | SET DEFAULT | SET NULL}]
```

- *numeCampCheieStraina* trebuie să fie indexat în tabela curentă, iar *numeCampCheiPrimara* trebuie să fie cheie primară în *numeTabela*. Câmpurile care intră în componenţa *numeCampCheieStraina* şi *numeCampCheiPrimara* trebuie sa fie compatibile.

Opţional, putem preciza acţiunea care va avea loc în cazul ştergerii (ON DELETE) sau modificării (ON UPDATE) câmpului ce este referit. Dacă lipsesc, pentru ambele este NO ACTION, ceea ce înseamnă că nu va avea loc nici o acţiune, celelalte variante având următoarele semnificaţii:

- CASCADE – în cazul ştergerii unei înregistrări, toate referinţele din tabelă către ea vor fi şterse, iar în cazul modificării, vor fi actualizate corespunzător;
- SET DEFAULT – setează valoarea implicită în cazul ştergerii sau modificării (câmpul trebuie să aibă o valoare implicită);
- SET NULL – stabileşte valoarea nulă în cazul ştergerii sau modificării (câmpul nu trebuie definit *NOT NULL*).

Aşadar, vom crea două chei străine în tabela *note*, una pentru câmpul `cursID`, referind câmpul `id` din cadrul tabelei *cursuri*, care este o cheie primară, alta pentru `studentID`, ce va referi câmpul `id` din cadrul tabelei *studenti*, de asemenea cheie primară. Presupunem că avem deja create tabele cu informaţiile iniţiale:

```
mysql> SELECT * FROM studenti;
+----+----------+---------+
| id | nume     | prenume |
+----+----------+---------+
|  1 | Bahmata  | Sorin   |
|  3 | Bulancea | Catalin |
+----+----------+---------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM cursuri;
+----+-----------------+---------+
| id | nume            | credite |
+----+-----------------+---------+
|  1 | Tehnologii WEB  |       6 |
|  2 | Programare JAVA |       6 |
|  4 | Baze de date    |       5 |
+----+-----------------+---------+
3 rows in set (0.00 sec)

mysql> SELECT * FROM note;
+--------+-----------+------+
| cursID | studentID | nota |
+--------+-----------+------+
|      2 |         1 |   10 |
|      1 |         3 |   10 |
|      1 |         1 |    9 |
|      4 |         3 |   10 |
+--------+-----------+------+
4 rows in set (0.00 sec)

mysql>
```

Informaţiile din cele trei tabele

Reamintim faptul că aceste tabele au fost create cu tipul *MyISAM*, tipul implicit folosit de MySQL. Pentru a schimba tipul, vom recurge la comanda ALTER TABLE, comandă permiţând modificarea structurii unei tabele după ce aceasta a fost creată. Astfel, primul lucru pe care îl vom face este schimbarea tipului tabelelor, din *MyISAM* în *InnoDB*:

```
mysql> ALTER TABLE studenti TYPE=INNODB;
Query OK, 2 rows affected (0.20 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE cursuri TYPE=INNODB;
Query OK, 3 rows affected (0.06 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE note TYPE=INNODB;
Query OK, 4 rows affected (0.06 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql>
```

Schimbarea tipului celor trei tabele

Urmează crearea celor două chei străine: câmpul `studentID` din tabela note, care va referi câmpul `id` din tabela *studenti* şi câmpul `cursID` referind câmpul `id` din tabela *cursuri*. Cum deja ambele câmpuri sunt indexate în cele trei tabele, vom adăuga doar cheile străine pentru tabela *note*, recurgând tot la comanda ALTER TABLE, cu opţiunea CASCADE atât pentru ştergere, cât şi pentru modificare:

```
mysql> ALTER TABLE note
    -> ADD FOREIGN KEY (studentID) REFERENCES studenti(id)
    -> ON DELETE CASCADE
    -> ON UPDATE CASCADE;
Query OK, 4 rows affected (0.11 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE note
    -> ADD FOREIGN KEY (cursID) REFERENCES cursuri(id)
    -> ON DELETE CASCADE
    -> ON UPDATE CASCADE;
Query OK, 4 rows affected (0.13 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql>
```

Crearea cheilor străine

În cazul în care primiţi un mesaj de eroare de genul:

```
ERORR 1005: Can't create table '.\test\#sql-9dc_1.frm'
(errno 50)
```

verificaţi consistenţa datelor din cadrul celor două tabele (informaţiile din câmpurile `cursID` şi `studentID` trebuie să existe în tabelele părinte). Observaţi efectul următoarelor comenzi:

```
mysql> UPDATE studenti SET id=5 WHERE id=1;
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> DELETE FROM cursuri WHERE id=4;
Query OK, 1 row affected (0.03 sec)

mysql> SELECT * FROM studenti;
+----+----------+---------+
| id | nume     | prenume |
+----+----------+---------+
|  3 | Bulancea | Catalin |
|  5 | Bahmata  | Sorin   |
+----+----------+---------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM cursuri;
+----+-----------------+---------+
| id | nume            | credite |
+----+-----------------+---------+
|  1 | Tehnologii WEB  |       6 |
|  2 | Programare JAVA |       6 |
+----+-----------------+---------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM note;
+--------+-----------+------+
| cursID | studentID | nota |
+--------+-----------+------+
|      2 |         5 |   10 |
|      1 |         3 |   10 |
|      1 |         5 |    9 |
+--------+-----------+------+
3 rows in set (0.00 sec)

mysql> INSERT INTO note VALUES (1,1,9);
ERROR 1216: Cannot add or update a child row: a foreign key constraint fails
mysql>
```

Modificările sau ştergerile din tabelele părinte sunt actualizate în tabela *note* datorită acţiunii de „cascadă" definite în cadrul cheii străine, iar inserarea de noi înregistrări trebuie să respecte relaţiile *părinte-copil*, păstrându-se astfel integritatea datelor

Înainte de a trece mai departe, trebuie menţionat faptul că aceşti paşi au fost evidenţiaţi pentru a înţelege avantajul cheilor străine, dar definirea lor se poate face direct la crearea tabelelor. În acest caz, considerăm următorul exemplu:

```
mysql> CREATE TABLE parinte
    -> (
    -> campParinte1 INT NOT NULL,
    -> campParinte2 VARCHAR(20) NOT NULL,
    -> PRIMARY KEY (campParinte1,campParinte2)
    -> ) TYPE=INNODB;
Query OK, 0 rows affected (0.06 sec)

mysql> CREATE TABLE copil
    -> (
    -> campCopil1 INT,
    -> campCopil2 VARCHAR(20),
    -> INDEX (campCopil1,campCopil2),
    -> FOREIGN KEY (campCopil1,campCopil2) REFERENCES parinte(campParinte1,campParinte2)
    -> ON DELETE CASCADE
    -> ) TYPE=INNODB;
Query OK, 0 rows affected (0.05 sec)

mysql>
```

Definirea cheii străine în momentul creării tabelei *copil*

Atunci când se doreşte ştergerea cheii străine, trebuie specificat numărul cheii străine, număr generat de serverul MySQL. Ştergerea este utilă mai ales când se doreşte modificarea acţiunilor referinţei, caz în care trebuie ştearsă vechea cheie străină şi definită o nouă cheie, cu acelaşi câmpuri, dar cu acţiuni diferite. Pentru a afla numărul cheii străine, se foloseşte comanda SHOW CREATE TABLE, care afişează şi cheile străine, chiar dacă ele au fost sau nu definite ulterior creării tabelei.

În cazul nostru, această comandă este aplicată tabelei *note*:

```
| note  | CREATE TABLE `note` (
  `cursID` int(11) default NULL,
  `studentID` int(11) default NULL,
  `nota` int(2) default NULL,
  KEY `cursID` (`cursID`),
  KEY `studentID` (`studentID`),
  CONSTRAINT `0_563` FOREIGN KEY (`cursID`) REFERENCES `cursuri` (`id`) ON DELET
E CASCADE ON UPDATE CASCADE,
  CONSTRAINT `0_567` FOREIGN KEY (`studentID`) REFERENCES `studenti` (`id`) ON D
ELETE CASCADE ON UPDATE CASCADE
) TYPE=InnoDB |
```

Afişarea modului de creare a tabelei *note*. Denumirile tabelelor, câmpurilor, precum şi numărul cheilor străine sunt încadrate de caracterul „`", iar definiţia cheilor străine este precedată de cuvântul „CONSTRAINT"

Ştergerea cheii străine, ca şi eliminarea oricărui index, se realizează tot cu ALTER TABLE, diferenţa între cele două fiind aceea că în cazul cheilor străine trebuie specificat numărul, iar în cazul indecşilor, numele:

```
mysql> ALTER TABLE note DROP FOREIGN KEY 0_563;
Query OK, 3 rows affected (0.09 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql>
```

Ştergerea cheii străine precizând numărul afişat de comanda „SHOW CREATE TABLE". Utilizatorul poate să îşi redefinească cheia străină urmând paşii menţionaţi, eventual cu alte acţiuni la ştergere şi/sau modificare

Menţionăm faptul că acţiunile pentru modificare sau ştergere nu trebuie să fie tot timpul CASCADE. De exemplu, există situaţii în care se doreşte a fi setată valoarea nulă sau implicită (în general, valoarea implicită este o valoare care nu apare în tabela părinte) pentru un anumit câmp, cum ar fi cazul în care câmpul referinţă este şters şi se doreşte a se arăta faptul că anumite înregistrări sunt „orfane". De asemenea, un câmp indexat poate avea referinţe către mai multe câmpuri, astfel selectându-se doar valorile comune câmpurilor referite.

3. Descrierea metadatelor

3.1 Introducere

În următorul pas al evoluției Web-ului, denumit *Web-ul semantic*, vaste cantități de resurse de informații (date, documente, programe) vor fi disponibile alături de diverse tipuri de informații descriptive, adică *metadate* (date despre date). O mai bună cunoaștere a înțelesului, folosirii, accesibilității sau calității resurselor Web va înlesni considerabil procesarea automată a conținutului/serviciilor Web. Web-ul semantic va fi pentru calculatoare ceea ce reprezintă Web-ul actual pentru noi, oamenii. Web-ul semantic are drept scop automatizarea întregului proces de descoperire, căutare și integrare a informațiilor. Programele vor fi capabile în mod autonom să găsească și să navigheze printre resursele Web. Acest aspect deschide o serie de posibilități extraordinare pentru viitorul apropiat și mai îndepărtat.

Pentru a reuși acest lucru, Web-ul semantic se bazează pe un model de date nou și mult mai general, *RDF* (*Resource Description Framework*). Modelul RDF constă într-un limbaj care permite crearea și schimbul metadatelor despre resurse ca orice date Web. Este destinat în particular reprezentării metadatelor privitoare la resursele Web, cum ar fi titlul, autorul și data ultimei modificări a unei pagini Web sau planul disponibil al unei resurse partajate. Totuși, prin generalizarea conceptului de „resursă Web", RDF poate fi folosit pentru a reprezenta informații despre lucruri ce pot fi *identificate* pe Web, chiar dacă ele nu pot fi direct *regăsite* pe Web. Una dintre caracteristicile specifice ale modelului RDF este abilitatea de a suprapune o serie de descrieri pentru aceeași resursă Web într-o varietate de contexte ale aplicației: publicitate, recomandare, *copyright* etc.

3.2 RDF: privire de ansamblu

RDF se bazează pe ideea că lucrurile care trebuie descrise au *proprietăți* având anumite valori și că resursele pot fi descrise prin afirmații, similare celei de mai jos, care specifică acele proprietăți și valori. RDF folosește o terminologie specială pentru fiecare parte a afirmațiilor. Astfel, partea care identifică lucrul despre care se vorbește în afirmație reprezintă *subiectul*. Partea care referă proprietatea subiectului specificată în afirmație reprezintă *predicatul*, iar cea care identifică valoarea proprietății reprezintă *obiectul*. Astfel, pornind de la afirmația:

http://students.infoiasi.ro/~sib/index.php are *titlul*
cu valoarea *Bine ați venit*

determinăm că:

- subiectul este *http://students.infoiasi.ro/~sib/index.php*;

- predicatul este cuvântul *„titlul"*;
- obiectul este grupul de cuvinte *„Bine ati venit"*.

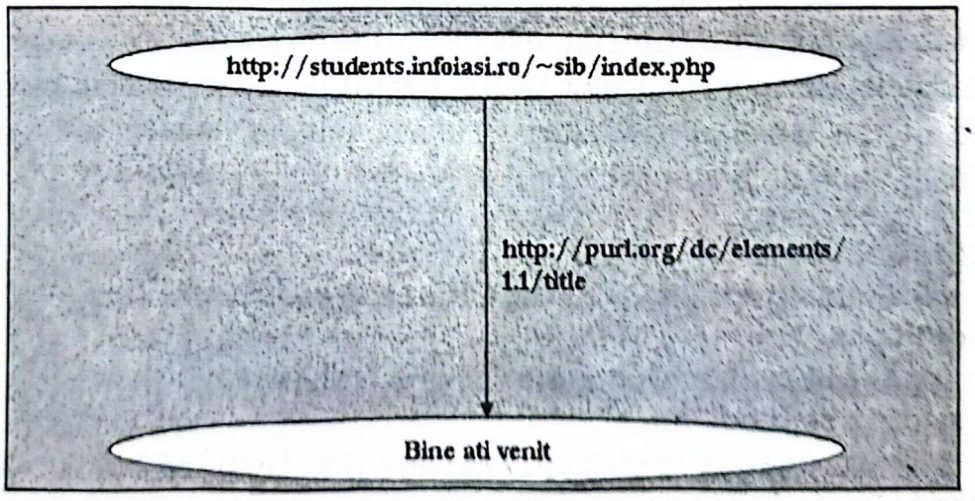
Spaţiul WWW oferă o metodă de identificare – URL-ul (*Uniform Resource Locator*) – referind o resursă Web prin modul de accesare a acesteia. Totuşi, este important a prelua informaţii şi despre alte lucruri care, spre deosebire de paginile Web, nu au URL-uri. Web-ul pune la dispoziţie o metodă mai generală pentru acest caz, numită *URI* (*Uniform Resource Identifier*). URL-urile sunt un caz particular al URI-urilor. Toate URI-urile au proprietatea că diferite organizaţii sau persoane pot să le creeze independent şi pot fi folosite pentru a identifica anumite lucruri. Nu se limitează la a identifica numai resurse de pe Web, aici putem include şi oameni, corporaţii, cărţile dintr-o librărie sau chiar concepte abstracte care fizic nu există, cum ar fi conceptul de *titlu*.

Datorită acestei generalităţi, RDF foloseşte URI-uri ca metodă de bază pentru a identifica subiectele, predicatele şi obiectele din cadrul afirmaţiilor. Mai precis, se folosesc *referinţele URI* (*URIref*). Un *URIref* este un URI cu un *identificator de fragment*, opţional, adăugat la sfârşit. De exemplu, referinţa URI *http://students.infoiasi.ro/~sib/index.php#sectiunea2* constă din URI-ul *http://students.infoiasi.ro/~sib/index.php* şi, separat prin caracterul „#", identificatorul de fragment *sectiunea2*. În acest fel, prin RDF putem descrie aproape orice lucru şi, de asemenea, formula relaţii între aceste lucruri.

Drept exemplu, afirmaţia precedentă ar avea ca:

- subiect: *http://students.infoiasi.ro/~sib/index.php*;
- predicat: *http://purl.org/dc/elements/1.1/title*;
- obiect: *Bine ati venit.*

Reprezentarea acestei afirmaţii RDF se face cu ajutorul grafurilor orientate, în care subiectul şi obiectul sunt noduri, iar predicatul reprezintă muchia orientată de la subiect la predicat:



Graful RDF

RDF oferă o sintaxă XML pentru a reproduce grafurile RDF, denumită RDF/XML. Ideea de bază a sintaxei o vom ilustra cu ajutorul exemplelor. Revenind la afirmaţia noastră, sintaxa este următoarea:

```
<?xml version="1.0"?>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-
ns#"
   xmlns:dc="http://www.purl.org/dc/1.1">

 <rdf:Description
   rdf:about="http://students.infoiasi.ro/~sib/
index.php">
   <dc:title>Bine ati venit</dc:title>
 </rdf:Description>

</rdf:RDF>
```

Prima linie reprezintă declaraţia XML, indicând faptul că acest document este un document XML. Următoarea linie începe cu elementul `rdf:RDF`, specificând faptul că documentul va include un conţinut RDF. Mai putem observa că există o declarare de spaţiu de nume, prefixat cu `rdf`. Această declaraţie specifică faptul că toate *tag*-urile prefixate cu `rdf` sunt parte a spaţiului de nume identificat prin *URIref*-ul *http://www.w3.org/1999/02/22-rdf-syntax-ns#* (acest *URIref* este folosit pentru a referi termeni ai vocabularului RDF). Desigur, pot fi indicate mai multe spaţii de nume. Elementul `rdf:Description` specifică începutul unei descrieri a unei resurse şi identifică resursa despre care se vorbeşte în afirmaţie folosind atributul `rdf:about` pentru a indica *URIref*-ul *subiectului*. Urmează *predicatul* şi *obiectul*, cuprinşi în cadrul elementului `dc:title`, sub element al elementului `rdf:Description`, ceea ce înseamnă că *predicatul* şi *obiectul* se aplică resursei specificate de atributul `rdf:about`. *Tag*-ul `dc:title`, aparţine specificaţiei *DCMI* (*Dublin Core Metadata Initiative*), care pune la dispoziţie un set de elemente referitoare la metadatele asociate resurselor Web – pentru detalii, consultaţi bibliografia.

Evident, pe lângă proprietatea `title` a paginii, putem adăuga şi alte proprietăţi, cum ar fi autorul ei, toate acestea fiind grupate:

```
<rdf:Description
   rdf:about="http://students.infoiasi.ro/~sib/
index.php">
   <dc:title>Bine ati venit</dc:title>
   <dc:creator
     rdf:resource="http://students.infoiasi.ro/~sib"/>
</rdf:Description>
```

În acest exemplu, elementul `dc:creator` reprezintă o proprietate a cărei valoare este o altă resursă. Dacă referinţa URI a acestei resurse ar fi scrisă ca un simplu text între *tag*-urile de început şi sfârşit, în aceeaşi manieră ca şi alte texte în alte elemente (cazul `dc:title`), acest lucru ar spune că valoarea elementului `dc:creator` este literalul *„http://students.infoiasi.ro/students/~sib"*, fără a se interpreta resursa identificată de acest literal ca fiind un *URIref*. Pentru a evidenţia acest lucru, valoarea proprietăţii este specificată prin atributul `rdf:resource`.

Situaţiile în care este nevoie de a descrie un grup de lucruri (de exemplu, pentru a afirma că o carte a fost scrisă de mai mulţi autori sau a specifica lista studenţilor înscrişi la un anumit curs) sunt rezolvate prin punerea la dispoziţie a unui set de tipuri şi proprietăţi predefinite. Un *container* este o resursă conţinând lucruri, denumite *membri*, ce pot fi alte resurse sau literali. Există trei tipuri de containere în RDF:

- `rdf:Bag` – reprezintă un grup de resurse ce poate avea şi elemente duplicate, în care ordinea nu este importantă;
- `rdf:Seq` – similar ca şi în cazul *rdf:Bag*, numai că ordinea este importantă;
- `rdf:Alt` – reprezintă un grup de resurse ce sunt alternative (tipic pentru proprietăţi ce pot avea o singură valoare dintr-o mulţime de valori).

Astfel, pentru a reprezenta afirmaţia „Cursul *Tehnologii Web* are ca studenţi persoanele Bahmătă Sorin, Bulancea Cătălin şi Puiu Daniel", avem următoarea sintaxă (spaţiul de nume „s" specifică un vocabular propriu, privitor la studenţi – aici, folosim elementul `<students>`):

```
<?xml version="1.0"?>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:s="http://students.infoiasi.ro/vocab#">

 <rdf:Description rdf:about=
   "http://thor.info.uaic.ro/~busaco/teach/courses/web">
   <s:students>
    <rdf:Bag>
       <rdf:li
       rdf:resource="http://students.infoiasi.ro/~sib"/>
       <rdf:li
       rdf:resource="http://www3.infoiasi.ro/
        ~catalinb"/>
       <rdf:li rdf:resource="mailto:dootnot@yahoo.com"/>
    </rdf:Bag>
   </s:students>
 </rdf:Description>
</rdf:RDF>
```

4. Descrierea aplicației *DataR*

4.1 Scopul aplicației

Aplicația *DataR* generează reprezentări RDF, denumite *RelDF*, care descriu structura bazelor de date relaționale MySQL, incluzând aici și relațiile dintre tabelele din cadrul acestora. Totodată, se permite și modelarea relațiilor între tabele (crearea, ștergerea și modificarea lor). Prin intermediul aplicației, vă puteți conecta la un server MySQL specificând adresa serverului (implicit va fi adresa IP a utilizatorului), iar opțional, numele și parola pentru autentificare. În cazul în care conexiunea nu s-a putut efectua, veți fi înștiințat despre motivul eșuării conexiuni, oferindu-vă posibilitatea de a vă reconecta.

În situația în care conexiunea la server s-a efectuat cu succes, aveți posibilitatea să realizați următoarele operații:

- vizualizarea bazelor de date la care aveți acces, precum și a tabelelor din cadrul acestora;
- introducerea/modificarea/ștergerea înregistrărilor din cadrul tabelelor pe baza relațiilor definite;
- crearea/ștergerea indecșilor din cadrul tabelelor;
- crearea/ștergerea/modificarea/vizualizarea relațiilor dintre tabele;
- adăugarea/ștergerea informațiilor despre bazele de date, din cadrul fișierului *RelDF*.

4.2 Modularizarea

Aplicația este dezvoltată pe două direcții. Prima direcție a fost menționată în cadrul secțiunii precedente. Cea de-a doua direcție este, așa cum am pomenit la începutul acestui capitol, de a interacționa cu aplicația *RelVis*, aplicație care prelucrează fișierele *RelDF* transmise și generează fișiere SVG pentru reprezentarea grafică a modelului relațional. Aplicația *RelVis* va fi detaliată în cadrul capitolului următor.

Pornind de la aceste direcții, lista fișierelor folosite de *DataR* este următoarea:

- *answerAtLogin.php* – este scriptul PHP folosit de aplicația *RelVis* pentru a afla dacă a reușit conexiunea cu serverul MySQL. Parametrii necesari sunt adresa serverului, numele și parola utilizatorului;
- *commonDB.php* – conține funcțiile de conectare la serverul MySQL și de afișare a erorilor MySQL;
- *createIDX.php* – script care creează indecși pentru tabele. Necesită numele tabelei, tipul de index (cheie primară, cheie unică sau index obișnuit) și numărul de câmpuri ce vor crea indexul;

- *db.php* – responsabil cu afişarea bazelor de date la care are acces utilizatorul. Prin selectarea unei baze de date, care devine astfel baza de date curentă, se actualizează fişierul *table.php*;
- *deleteRel.php* – permite utilizatorului să şteargă relaţii definite între două tabele, având drept parametri numele celor două tabele;
- *detailsRel.php* – afişează detaliile despre relaţiile dintre două tabele, parametrii fiind aceiaşi ca şi în cazul precedent;
- *editTB.php* – este responsabil cu adăugarea, ştergerea şi modificarea înregistrărilor dintr-o tabelă. Necesită numele tabelei;
- *fullRDF.php* – script folosit de aplicaţia *RelVis*, furnizând structura tuturor bazelor de date la care are acces un anumit utilizator printr-un fişier *RelDF*. Este accesat doar dacă rezultatul returnat de *answerAtLogin.php* este pozitiv, parametrii acestuia fiind aceiaşi ca şi în cazul ultimului;
- *getPrimaryKeys.php* – determină toate cheile primare din baza de date curentă. Acest script este folosit la crearea relaţiilor între tabele;
- *getRDF.php* – script invocat de aplicaţia *RelVis*, responsabil cu furnizarea înregistrărilor pentru o anumită tabelă, sortate după un câmp din cadrul structurii sale. Pe lângă adresa serverului, numele şi parola utilizatorului, acest fişier are nevoie şi de numele bazei de date, tabelei din cadrul ei şi al câmpului după care se doreşte sortarea;
- *index.php* – fişierul principal al aplicaţiei. Conţine detaliile oferite de fişierele *db.php* şi *table.php*. Permite şi deconectarea utilizatorului;
- *login.php* – responsabil cu conectarea la serverul de baze de date;
- *set.php* – setează variabilele de sesiune ce oferă informaţiile despre serverul MySQL;
- *table.php* – vizualizează structura bazei de date curente (tabelele şi relaţiile dintre acestea) şi generează fişierele *RelDF*;
- *viewtable.php* – afişează structura unei tabele (câmpuri şi indecşi) şi înglobează funcţionalităţile oferite de fişierele *edit.php* şi *create.php*. De asemenea, permite crearea de relaţii între tabela curentă şi celelalte tabele din baza de date utilizată.

Evident, pe lângă aceste fişiere, pot fi adăugate fişiere CSS, plus imaginile folosite pentru realizarea interfeţei aplicaţiei, eventual fişiere HTML cu rolul de a oferi informaţii de ajutor privitoare la anumiţi termeni (indecşi, chei străine, crearea cheilor străine etc.).

4.3 Detalii de implementare

Înainte de a trece la sarcinile aplicaţiei, vom descrie funcţionalităţile unora dintre fişierele menţionate în secţiunea precedentă.

Așa cum am precizat, scriptul *commonDB.php* conține funcțiile de conectare la serverul de baze de date și de afișare a erorilor de tip MySQL:

```
<?php
$ERRNO = ''; //numarul erorii
$ERROR = ''; //eroarea
/*
* $host     - adresa serverului
* $username - numele utilizatorului
* $password - parola utilizatorului
*/
function MySQLConnect($host, $username, $password)
{
    global $ERRNO, $ERROR;

    $conn = @mysql_connect($host, $username, $password);
    if(!$link_id)
    {
        $ERRNO = mysql_errno();
        $ERROR = mysql_error();
        return 0;
    }
    return $conn;
}
/*
* $showAlert - daca mesajul de eroare va fi afisat cu
*              functia alert() din JavaScript sau nu
*/
function MySQLError($showAlert)
{
    global $ERRNO, $ERROR;

    //daca nu au fost setate
    if(empty($ERROR) || empty($ERRNO))
    {
        $ERRNO = mysql_errno();
        $ERROR = mysql_error();
    }

    if($showAlert)
    {
        ?>
        <script type="text/javascript">
        alert("Error number: <? echo $ERROR; ?>");
        </script>
        <?
        return;
    }
```

```
        return "<b>Error number $ERRNO:</b>
                <br/>$ERROR";
}
?>
```

Deoarece fiecare operaţie asupra tabelelor necesită o conexiune la serverul MySQL, aceste funcţii vor fi folosite de majoritatea scripturilor şi, pentru a nu le scrie de fiecare dată în fiecare fişier în parte, le vom descrie în cadrul acestui fişier şi îl vom include cu ajutorul funcţiei `include()`:

```
<?php
    include("../commonDB.php");
?>
```

Fişierul *login.php* preia de la utilizator adresa serverului, numele şi parola utilizatorului şi le trimite către *set.php*, care setează aceste variabile drept variabile de sesiune. Ele trebuie să persiste pe toată durata aplicaţiei sau până când utilizatorul doreşte să se deconecteze. Structura fişierului este următoarea:

```
<?php
    session_start();

    if(isset($_SESSION['host']) ||
       isset($_SESSION['username']) ||
       isset($_SESSION['password']))
    {
       unset($_SESSION['host'],$_SESSION['username'],
             $_SESSION['password']);
       session_destroy();
    }//stergem vechile valori

    if (($_POST['host'] == '127.0.0.1') ||
        ($_POST['host'] == ''))//server local
           $_SESSION['host'] = 'localhost';
    else
           $_SESSION['host'] = $_POST['host'];

    $_SESSION['username'] = $_POST['username'];

    if($_POST['password'] != '')//parola trebuie criptata
    {
       $_SESSION['userpassword'] =
           base64_encode($_POST['password']);
    }
    header('Location: index.php');
?>
```

Cele mai importante sarcini în implementarea aplicaţiei sunt modelarea relaţiilor între tabele, generarea fişierelor *RelDF* şi interacţiunea cu aplicaţia *RelVis*.

Relaţiile dintre tabele

Pentru a determina relaţiile dintre tabele, mai exact cheile străine a unei tabele, avem două metode: folosind comanda SHOW CREATE TABLE sau utilizând comanda SHOW TABLE STATUS.

Prima metodă a fost discutată mai devreme în cadrul acestui capitol. Cea de-a doua este mai săracă în informaţii, deoarece comanda oferă detalii despre toate tabelele din baza de date curentă, mai puţin numărul cheilor străine, esenţial în cazul în care dorim să le ştergem sau să le modificăm. Pornind de la aceleaşi exemple de tabele, comanda SHOW TABLE STATUS are următoarea execuţie posibilă:

```
mysql> SHOW TABLE STATUS;
+----------+--------+------------+------+----------------+-------------+-------
---------+-------------+-----------+----------------+-------------+-----------
-+------------+----------------+-----------------------------------------------
-----------------------------------------------------------------------------
---+
| Name     | Type   | Row_format | Rows | Avg_row_length | Data_length | Max_dat
a_length | Index_length | Data_free | Auto_increment | Create_time | Update_time
 | Check_time | Create_options | Comment

   |
+----------+--------+------------+------+----------------+-------------+-------
---------+-------------+-----------+----------------+-------------+-----------
-+------------+----------------+-----------------------------------------------
-----------------------------------------------------------------------------
---+
| cursuri  | InnoDB | Dynamic    |    0 |              0 |       16384 |
    NULL |            0 |         0 |           NULL | NULL        | NULL
 | NULL       |                | InnoDB free: 10240 kB

   |
| note     | InnoDB | Fixed      |    0 |              0 |       16384 |
    NULL |        32768 |         0 |           NULL | NULL        | NULL
 | NULL       |                | InnoDB free: 10240 kB; (cursID) REFER test /cur
suri(id) ON UPDATE CASCADE; (studentID) REFER test /studenti(id) ON UPDATE CASCA
DE |
| studenti | InnoDB | Dynamic    |    0 |              0 |       16384 |
    NULL |            0 |         0 |           NULL | NULL        | NULL
 | NULL       |                | InnoDB free: 10240 kB

   |
+----------+--------+------------+------+----------------+-------------+-------
---------+-------------+-----------+----------------+-------------+-----------
-+------------+----------------+-----------------------------------------------
-----------------------------------------------------------------------------
---+
3 rows in set (0.02 sec)

mysql>
```

Informaţiile despre tabelele din baza de date curentă

Se observă în dreptul tabelei *note*, la câmpul *Comment*, detalii despre cele două chei străine, însă lipseşte numărul cheii străine. Dacă doriţi doar aflarea relaţiilor, nu şi ştergerea/modificarea lor, atunci puteţi folosi una dintre aceste două posibilităţi.

Iată şi codul pentru aflarea relaţiilor dintre o anumită tabelă şi celelalte tabele:

```
/*
* $table       - tabela pentru care dorim sa aflam
                 relatiile
* $relations   - tablou ce va retine relatiile intre tabele
*/
function detRels($table, &$relations)
{
     $query  = "show create table $table";
     $result = mysql_query($query);

     $create     = mysql_fetch_row($result);
     $create[1] = str_replace("`", "", $create[1]);

     $array = explode(" CONSTRAINT ",$create[1]);
     if(count($array) != 1)
     {
          //are relatii, le adaugam in tablou
          //trecem peste primul element (de pe poz. 0)
          for($i = 1; $i < count($array); $i++)
          {
             if($i == (count($array)-1))
             {
                //se termina in " ) TYPE=... ", eliminam
                $temp = explode(") TYPE",$array[$i]);
                $array[$i] = $temp[0];
             }
             if($i != (count($array)-1))
             {
                $array[$i] =
                     substr( $array[$i], 0,
                             strlen($array[$i])-3);
             }
             else
                $array[$i] =
                     substr( $array[$i], 0,
                             strlen($array[$i])-1);
          //adaugam relatia
          $relations[] = $array[$i];}
     }
}
```

Să analizăm acest cod. Mai întâi, determinăm modul în care a fost creată tabela *$table* cu ajutorul funcţiei `mysql_query()`, apelând comanda SHOW

CREATE TABLE, rezultatul reținându-se în variabila `$create`. Așa cum am putut vedea, atunci când se apelează comanda SHOW CREATE TABLE, numele tabelelor, câmpurilor, precum și numărul cheii străine sunt încadrate de caracterul „`'`". Acest lucru este incomod în prelucrarea informațiilor primite, substituirea sa cu șirul vid făcându-se cu ajutorul funcției PHP predefinite `str_replace()`.

Dacă există chei străine, acestea sunt precedate de CONSTRAINT, caz în care vom determina grupurile de cuvinte despărțite de acest cuvânt cu ajutorul funcției `explode()`. Această funcție primește ca parametri două variabile, un text și un separator, ambele de tip *string*, și returnează un tablou al grupurilor de caractere din cadrul primei variabile, caractere delimitate de separator. Pentru a determina relațiile, ne interesează toate aceste grupuri de cuvinte, pe care le reținem în tabloul *$relations*, exceptând primul grup. Ultimul grup conține și tipul tabelei *$table*, deci îl eliminăm din cadrul grupului.

Cu alte cuvinte, *$relations* va reține pentru tabela *$table*, un tablou cu fiecare relație ce are drept tabela copil *$table*. Revenind la exemplele noastre, pentru tabela *note* se va reține următorul tablou:

```
{
0 => "0_563 FOREIGN KEY (cursID) REFERENCES cursuri(id)
ON DELETE CASCADE ON UPDATE CASCADE",
1 => "0_567 FOREIGN KEY (studentID) REFERENCES
cursuri(id)
ON DELETE CASCADE ON UPDATE CASCADE",
}
```

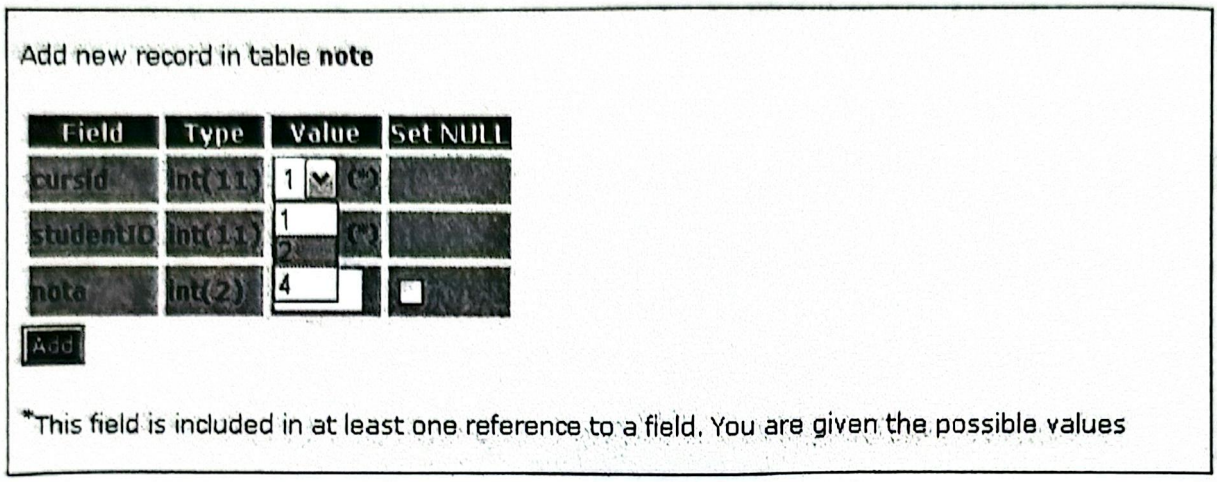
Aceste informații urmează a fi procesate pentru a obține detalii despre fiecare relație în parte. Pe baza acestor date, funcțiile de creare, de ștergere, și de modificare a relațiilor sunt următoarele:

```
/*
* $tbParent - tabela parinte
* $tbChild  - tabela copil
* $idx      - tabloul campurilor indexate din cadrul
*             $tbChild
* $prk      - tabloul campurilor ce alcatuiesc cheia
*             primara din cadrul $tbParent
* $opt      - actiunile la stergere/modificare
*/
function createRel($tbParent, $tbChild, $idx, $prk, $opt)
{
    //modificam tipul celor doua table
    $query = "ALTER TABLE $tbChild TYPE=INNODB";
    mysql_query($query);
```

```
    $query = "ALTER TABLE $tbParent TYPE=INNODB";
    mysql_query($query);

    $query = "ALTER TABLE $tbChild ADD FOREIGN KEY (";
    for($i = 0; $i < count($idx); $i++)
    {
        $query .= $idx[$i];
        if($i != count($idx) -1)
           $query .= ",";
    }

    $query .= ") REFERENCES $tbParent(";
    for($i = 0; $i < count($prk); $i++)
    {
        $query .= $prk[$i];
        if($i != count($prk) -1)
           $query .= ",";
    }

    $query .= ") $opt";
    mysql_query($query);
}

/*
* $tbChild  - tabela copil
* $nb       - numarul cheii straine
*/
function delRel($tbChild, $nb)
{
    $query = "ALTER TABLE $tbChild DROP FOREIGN KEY $nb";
    mysql_query($query);
}

/*
* $tbParent - tabela parinte
* $tbChild  - tabela copil
* $nb       - numarul cheii straine
* $idx      - tabloul campurilor indexate din cadrul
*             $tbChild
* $prk      - tabloul campurilor ce alcatuiesc cheia
*             primara din cadrul $tbParent
* $opt      - actiunile la stergere/modificare
*/

function updateRel($tbParent, $tbChild, $nb,
                   $idx, $prk, $opt) {
```

```
        //stergem vechea relatie
        delRel($tbChild, $nb);
        //o cream cu noile optiuni
        createRel(($tbParent, $tbChild, $idx, $prk, $opt);
    }
```

Atunci când dorim să adăugăm noi înregistrări în cadrul unei tabele, se verifică dacă datele respectă relaţiile *părinte-copil*, caz în care, dacă introducem în tabela copil valori ce nu există în tabela părinte, suntem atenţionaţi printr-un mesaj de eroare. Situaţiile de acest gen se dovedesc neplăcute, însă – prin intermediul aplicaţiei *DataR* – acestea sunt evitate:



În momentul introducerii unei noi înregistrări, se pot selecta doar înregistrările din tabela părinte

Deoarece un câmp poate referi mai multe câmpuri din diferite tabele părinte, se selectează doar valorile comune acestora, pentru a nu putea fi încălcate relaţiile. Pentru aceasta, vom determina câmpurile către care se face referinţă şi vom selecta doar valorile comune. Codul este următorul:

```
<?php
//...
$relations = array();
detRels($table,$relations);//determinam relatiile
$array = new array();//tablou ce va retine campurile
                     //referite de catre un anumit camp
                     //din cadrul tabelei $table
for($i=1;$i<count($relations);$i++)
{
    $rel = explode(" REFERENCES ", $relations[$i]);
    $temp = explode(" FOREIGN KEY ", $rel[0]);
    $rel = explode(" ON ", $rel[1]);

    $temp[1] = str_replace("(", "", $temp[1]);
    $temp[1] = str_replace(")", "", $temp[1]);
```

```
        $withTable = explode(" (", "", $rel[0]);

        $rel[0] = $withTable[1];
        $rel[0] = str_replace(")", "", $rel[0]);

        $temp = explode(", ", $temp[1]);

        $rel = explode(", ", $rel[0]);

        for($j = 0; $j < count($temp); $j++)
        {
            $array[$temp[$j]] .= withTable[0]. "." .
                                 $rel[$j] . " ";
        }
        //in acest moment, tabloul $array contine elemente
        //de genul:
        //numeTB1.numeCheiePr1, numeTB2.numeCheiePr2,...
}
?>
<p class="general">
Add new record in table <b><? echo $table; ?></b>
</p>
<form action="edit.php" method="post">
<table border="1">
<tr>
        <th>Field</th>
        <th>Type</th>
        <th>Value</th>
        <th>Set NULL</th>
</tr>
        <?
        //determinam structura tabelei
        $structure = mysql_query("desc $table");

        while($row = mysql_fetch_array($structure))
        {
            echo "<tr class='b'>";
            echo "<td>" . $row["Field"] . "</td>";
            echo "<td>" . $row["Type"] . "</td>";
            $parts = explode("(", $row["Type"]);
            $parts[1] = str_replace(")","", $parts[1]);
            if($array[$row["Field"]] != "")
            {
            //are relatie cu cel putin un camp
            ?>
            <td>
```

```
<select name="<? echo $row["Field"];?>">
<?php
//determinam coloanele si selectam valorile
//comune
$columns = explode(" ",
          $array[$row["Field"]]);
$query = "select " . $columns[0] . " from ";
for($i = 0; $i < count($columns) - 1; $i++)
{
   $tbl   = explode(".",$columns[$i]);
   $query = $query . $tbl[0];
   if($i != count($columns) - 2 )
         $query = $query. ", ";
}

if(count($columns) != 2)
{
   $query = $query. "where ";

   for($i = 1; $i < count($columns) - 1; $i++)
   {
         $query = $query.$columns[0].. " = " .
                        $columns[$i];
         if($i != count($columns) - 2 )
                  $query = $query . "and ";
   }
}
$result = @mysql_query($query);
if(mysql_num_rows($result) == 0)
{
?>
<!-- tratam erorile survenite -->
<script type="text/javascript">
alert("The field row <? echo $row['Field'];?>
       has more than one reference. " +
      "Possible no common records between the
       referenced fields. Could not add a new
       record.");
</script>
<?
//...
}
while($value = mysql_fetch_row($result))
{
?>
<option value="<? echo $value[0];?>">
<? echo $value[0];?>
```

```
            </option>
            <?php
            echo "</select>";
            }
?>
```

Generarea fişierelor *RelDF*

Vom descrie în continuare structura fişierelor *RelDF*. Elementul rădăcină este `rdf:RDF`, în cadrul lui definindu-se toate spaţiile de nume necesare:

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-sintax-ns#"
   xmlns:ns="http://www3.infoiasi.ro/~sib/ns"
   xmlns:dc="http://purl.org/dc/elements/1.1/">
```

Primul subelement al rădăcinii se ocupă de descrierea conexiunii la serverul MySQL: adresa, numele utilizatorului, parola (evident, criptată) şi bazele de date:

```
<rdf:Description rdf:about="server">
 <ns:host>localhost</ns:host>
 <ns:user>root</ns:user>
 <ns:password>WjJWMFpuSmxZV3Q1</ns:password>
 <ns:databases>
    <rdf:Bag>
      <rdf:li resource="mysql"/>
      <rdf:li resource="test"/>
    <!-- Alte baze de date -->
    </rdf:Bag>
 </ns:databases>
</rdf:Description>
```

În cadrul elementului `ns:databases`, se specifică bazele de date cu ajutorul elementului `rdf:li`, subelement al unui `rdf:Bag`. Următoarele elemente descriu, pe rând, aceste bazele de date:

```
<rdf:Description rdf:about="test">
 <ns:tables>
  <rdf:Bag>
   <rdf:li resource="test:cursuri"/>
   <rdf:li resource="test:note"/>
   <rdf:li resource="test:studenti"/>
   <!-- Alte tabele -->
  </rdf:Bag>
 </ns:tables>
</rdf:Description>
```

Ca şi în cazul precedent, tabelele sunt reprezentate printr-un element `rdf:Bag`. Denumirile tabelelor sunt precedate de denumirea bazei de date căreia îi aparţin, pentru a face distincţie între două tabele cu aceeaşi denumire, dar din baze de date diferite.

În continuare, se descrie fiecare tabelă în parte, şi anume: ID-ul tabelei, câmpurile (coloanele) tabelei, indecşii definiţi pentru aceasta şi relaţiile cu alte tabele din cadrul aceleiaşi baze de date. Dacă tabela în cauză nu are indecşi sau relaţii, elementele `ns:indexes` şi, respectiv, `ns:relations` vor conţine un `rdf:Bag` vid. În exemplul de faţă, între tabelele *copil* şi *parinte* din cadrul bazei de date *test* avem relaţii bazate pe cheia străină cu numărul 0_138, definită de la grupul (`copil.campCopil1`, `copil.campCopil2`) la grupul (`parinte.campParinte1`, `parinte.campParinte2`), cheie primară în tabela *parinte*:

```
<rdf:Description rdf:about="test:copil">
     <ns:id>3</ns:id>
      <ns:columns>
        <rdf:Seq>
          <rdf:li>
            <ns:column>
              <ns:name>campCopil1</ns:name>
              <dc:type size="11">int</dc:type>
            </ns:column>
          </rdf:li>
          <rdf:li>
            <ns:column>
              <ns:name>campCopil2</ns:name>
              <dc:type size="20">varchar</dc:type>
            </ns:column>
          </rdf:li>
        </rdf:Seq>
      </ns:columns>
      <ns:indexes>
        <rdf:Bag>
          <rdf:li>
            <ns:index ns:name="campCopil1"
                      ns:unique="no">
              <rdf:seq>
                <rdf:li>campCopil1</rdf:li>
                <rdf:li>campCopil2</rdf:li>
              </rdf:seq>
            </ns:index>
          </rdf:li>
        </rdf:Bag>
      </ns:indexes>
```

```
        <ns:relations>
          <rdf:Bag>
            <rdf:li>
              <ns:relation ns:constraint="0_138">
                <ns:index>
                  <rdf:Seq>
                    <rdf:li>campCopil1</rdf:li>
                    <rdf:li>campCopil2</rdf:li>
                  </rdf:Seq>
                </ns:index>
                <ns:reference
                  rdf:resource="test:parinte">
                  <rdf:Seq>
                    <rdf:li>campParinte1</rdf:li>
                    <rdf:li>campParinte2</rdf:li>
                  </rdf:Seq>
                  <ns:attributes> ON DELETE CASCADE
                  </ns:attributes>
                </ns:reference>
              </ns:relation>
            </rdf:li>
          </rdf:Bag>
        </ns:relations>
</rdf:Description>
```

Crearea fişierelor *RelDF* se poate realiza cu ajutorul funcţiilor DOM din limbajul PHP: se instanţiază un obiect de tip *DomDocument*, se adaugă/ elimină elemente şi se salvează fişierul *RelDF* rezultat. Denumirea fişierului coincide cu identificatorul de sesiune din PHP, dat de funcţia `session_id()`. S-a folosit această metodă pentru a nu apărea situaţia ca mai mulţi utilizatori să modifice simultan acelaşi fişier.

Pentru procesarea DOM a documentelor XML, PHP pune la dispoziţie funcţiile modulului sau extensiei *php_domxml.dll* (în Windows) şi *php_domxml.so* (în Linux). Pentru a crea sau modifica un document XML/ RDF, noua structură şi noul său conţinut trebuie mai întâi încărcate în memorie într-un arbore DOM. După finalizarea acestei acţiuni, pentru a salva arborele într-un fişier XML local, trebuie realizat un *dump* al acestuia în fişierul respectiv. Trebuie menţionat aici faptul că, dacă se încarcă arborele DOM al unui fişier XML deja creat şi acesta este modificat în memorie, *efectele modificării se vor observa şi în fişierul XML doar în cazul în care se apelează funcţia* `dump_file()`. Fiind o procesare bazată pe obiecte, toate variabilele folosite în cadrul procesării DOM sunt obiecte, având drept clasă de bază clasa *DomNode*.

Funcţiile pentru procesarea DOM din cadrul PHP sunt suficient de auto-descriptive. Vom oferi drept exemplu partea de cod pentru descrierea bazei de date *test* prezentate mai sus:

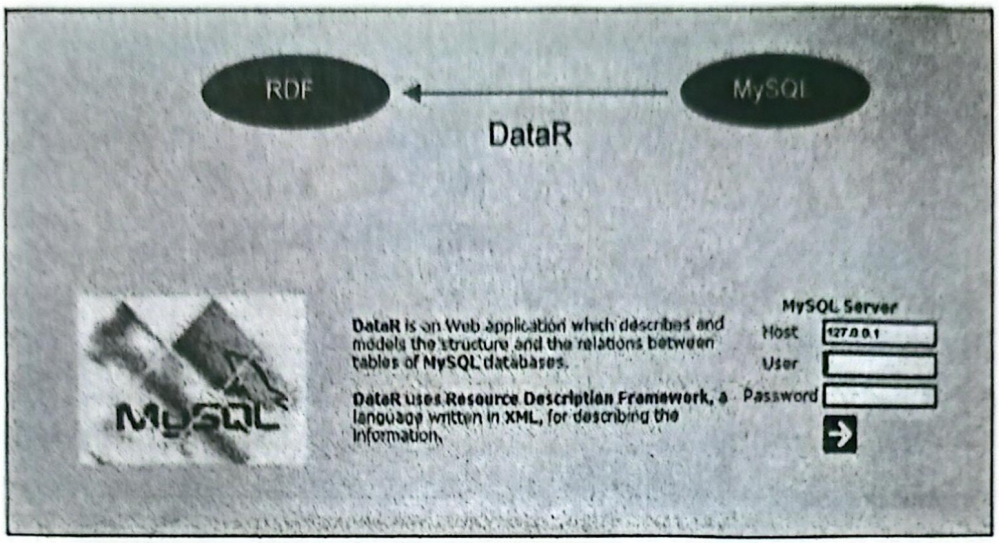
```
<?php
$doc = domxml_new_doc("1.0");
$root = $doc->create_element("rdf:RDF");
$root->set_attribute( "xmlns:rdf",
       "http://www.w3.org/1999/02/22-rdf-sintax-ns#");
$root->set_attribute( "xmlns:ns",
       "http://www3.infoiasi.ro/~sib/ns");
$root->set_attribute( "xmlns:dc",
       "http://purl.org/dc/elements/1.1/");
$root = $doc->append_child($root);
//...
$dbdesc = $doc->create_element("rdf:Description");
$dbdesc->set_attribute ("rdf:about", $crtDB);
$dbdesc = $root->append_child($dbdesc);
$tables = $doc->create_element("ns:tables");
$bag    = $doc->create_element("rdf:Bag");
$bag    = $tables->append_child($bag);
$tables = $dbdesc->append_child($tables);
$tbs = @mysql_list_tables($crtDB);
while($tb = mysql_fetch_row($tbs)
{
    $li = $doc->create_element("rdf:li");
    $li->set_attribute ("resource",
       $crtDB. ":" . $tb[0]);
    $bag->append_child($li);
}
//...
$doc->dump_file("/files/" . session_id(), false, true);
unset($doc);
?>
```

Interacţiunea cu aplicaţia *RelVis*

În cadrul interacţiunii cu *RelVIS*, se verifică dacă s-a putut realiza conexiunea la serverul de baze de date prin intermediul scriptului *answerAtLogin.php*, caz în care are loc interacţiunea între cele două aplicaţii. Spre deosebire de cele descrise mai sus, în acest caz, fişierele *RelDF* sunt create temporar de fiecare dată când aplicaţia *RelVIS* le solicită, ştergerea lor realizându-se după ce conţinutul lor a fost expediat pentru a putea fi procesat. Mai multe detalii vor furnizate în următorul capitol.

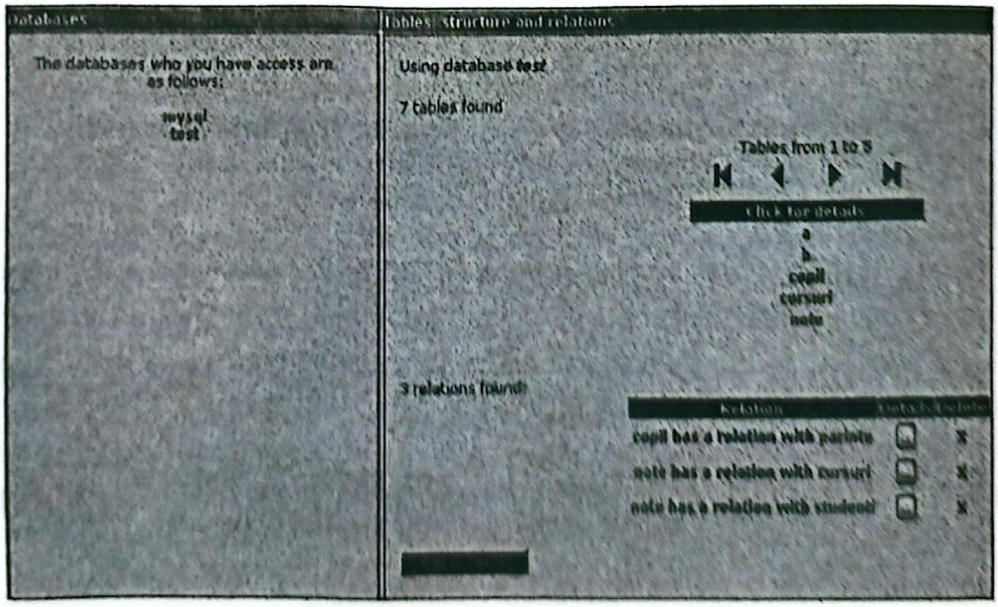
5. Manual de utilizare

În cadrul acestei secţiuni vom încerca să acoperim cât mai general modul de utilizare a aplicaţie *DataR*. Pentru aceasta, aveţi nevoie de lansarea fişierului *login.php* de pe adresa serverului Web unde este copiat codul sursă, acest lucru realizându-se cu ajutorul unui *browser* Web. Pe durata acestei secţiuni vom considera drept navigator *Internet Explorer*.



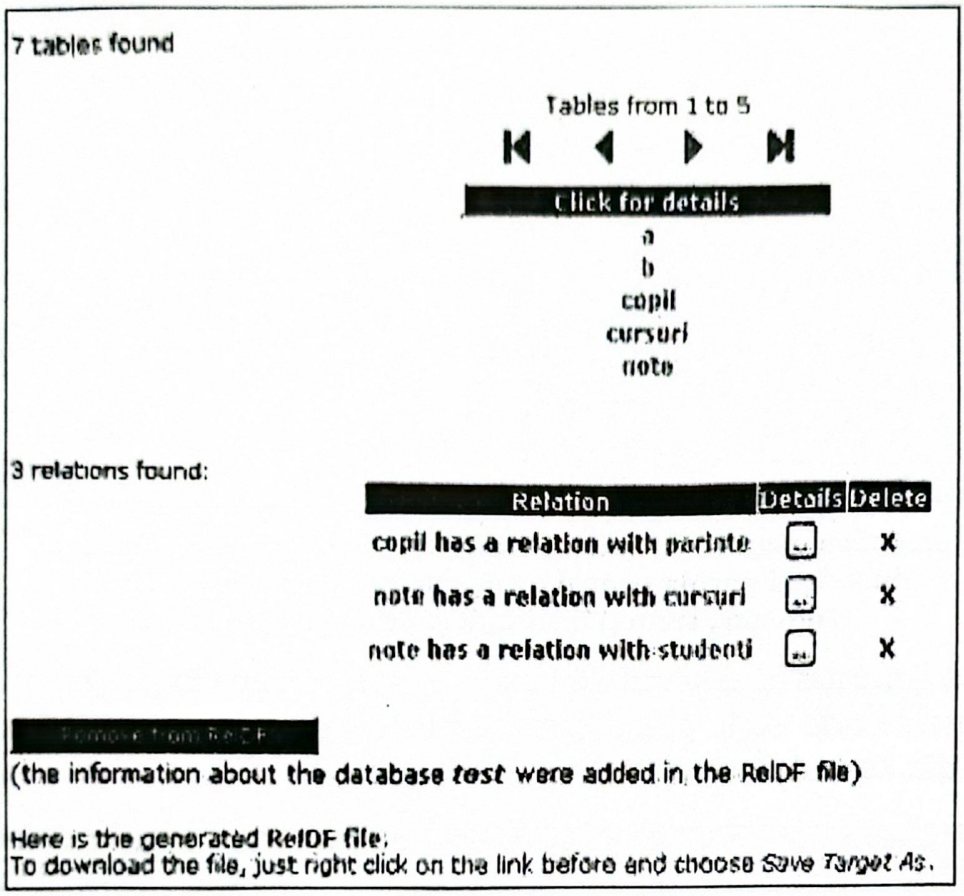
În cadrul paginii de autentificare trebuie să specificaţi adresa, numele şi parola pentru a vă conecta la un server MySQL. Totodată, sunteţi înştiinţat şi de scopul aplicaţiei

O dată conectat la serverul de baze de date, se trece în pagina principală a aplicaţiei, unde sunt prezentate bazele de date la care aveţi acces şi tabelele din cadrul fiecăreia:



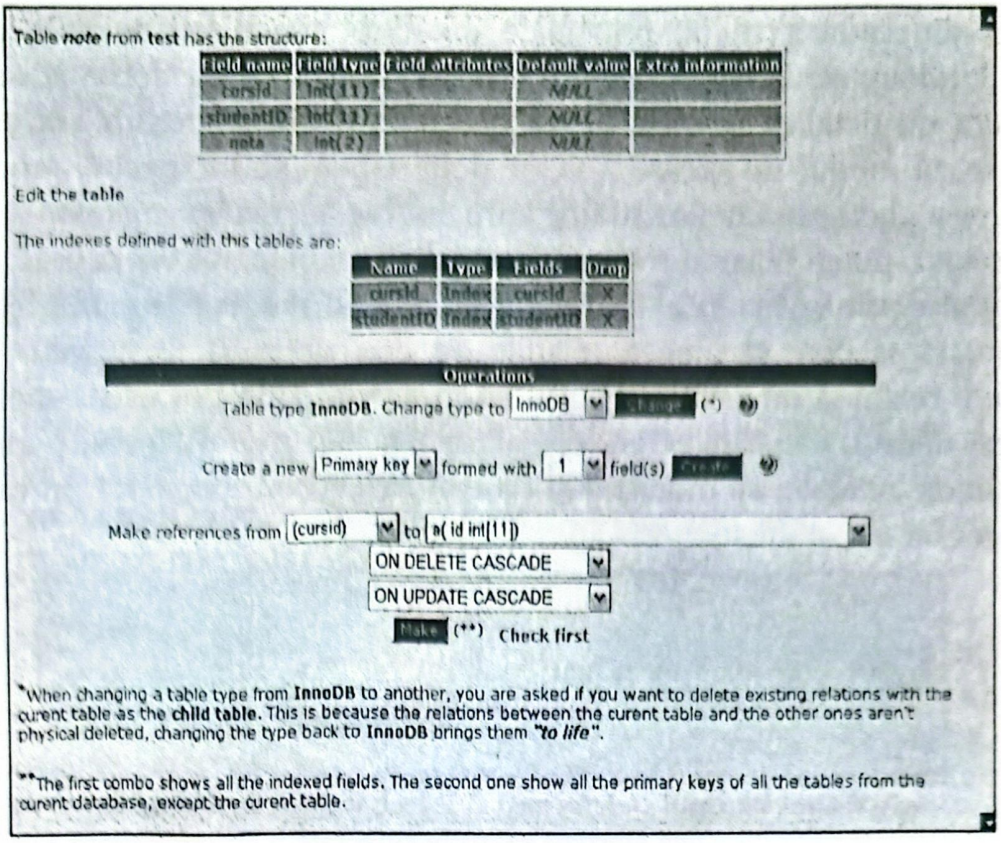
Pagina principală a aplicaţiei. În partea stângă a ecranului aveţi bazele de date, iar în dreapta – tabelele din baza de date curentă şi relaţiile dintre ele

Din cadrul paginii principale, puteţi să adăugaţi informaţiile bazei de date curente într-un fişier *RelDF* prin apăsarea butonului *Add to RDF*, adăugându-se informaţiile la fişier, dacă există, sau creând unul nou, dacă nu a fost deja creat. Va apărea şi o legătură către acest fişier. Desigur, se poate efectua şi operaţia inversă, de eliminare a informaţiilor bazei de date, doar dacă ele se găsesc în cadrul fişierului *RelDF*, prin intermediul butonului *Remove from RelDF*:

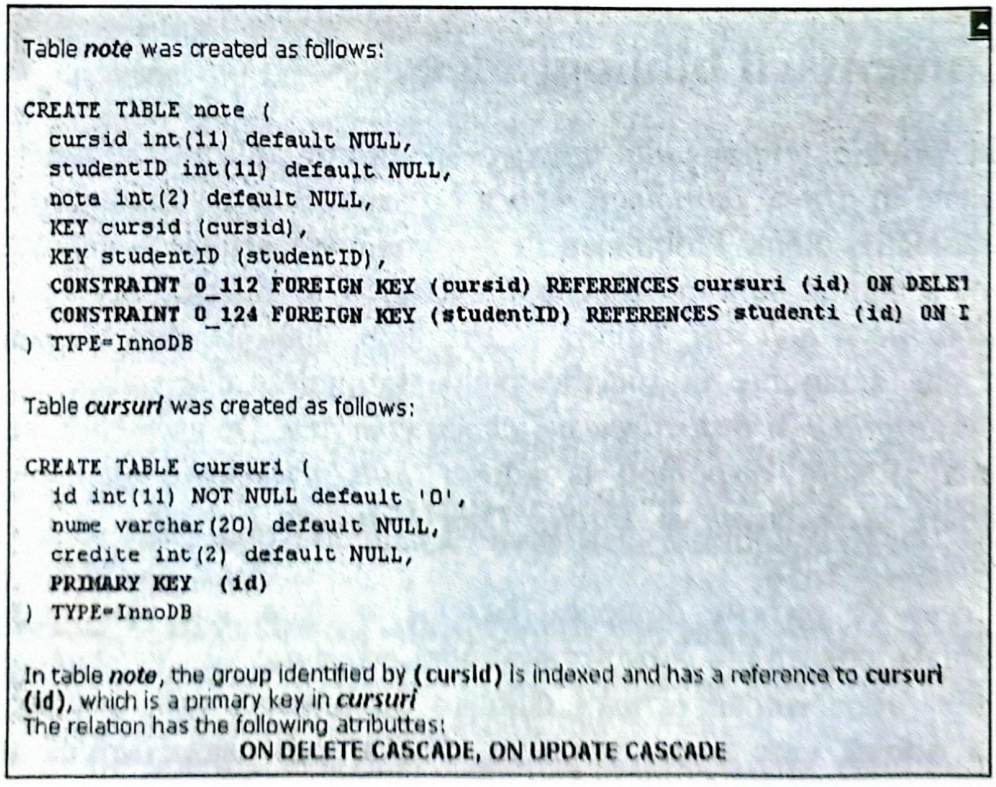


Adăugarea informaţiilor în fişier permite vizualizarea acestuia, dar şi eliminarea informaţiilor din cadrul lui

Selectând una dintre tabelele bazei de date curente, se va deschide o nouă fereastră, în care sunt prezentate datele despre tabela respectivă: câmpurile (denumire, tip, valori implicite etc.) şi indecşii definiţi (dacă există). Totodată, aveţi posibilitatea de a edita tabela, de a şterge sau crea noi indecşi, de a schimba tipul tabelei şi de a crea relaţii cu celelalte tabele din cadrul bazei de date curente. Dacă nu sunteţi familiarizaţi cu aceste noţiuni, există legături spre pagini Web care vă explică succint noţiunile respective:



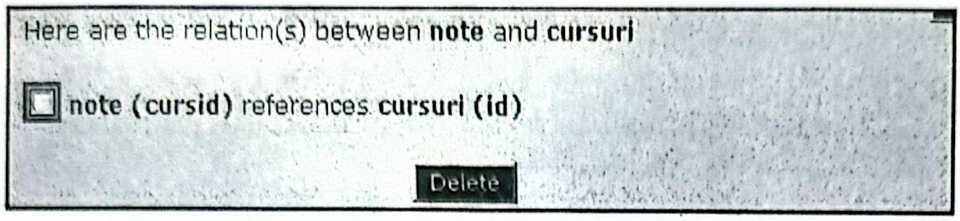
Detalii despre tabela selectată. Orice schimbare asupra tabelei conduce la modificarea fişierului *RelDF*, în cazul în care informaţiile despre baza de date curentă au fost adăugate. Detaliile despre fiecare operaţie sunt explicate prin legături către fişierele din cadrul directorului *help* al aplicaţiei



Detaliile privitoare la relaţia dintre două tabele

Tot din cadrul paginii principale puteţi să vizualizaţi detaliile despre relaţiile dintre două tabele. Pentru aceasta, alegeţi relaţia dorită şi selectaţi legătura de detalii (*Details*). Se va deschide o nouă fereastră, în care se precizează modul de creare a celor două tabele şi, în special, modul de definire a cheii sau cheilor străine care au dus la crearea relaţiilor.

Desigur, puteţi chiar şi şterge relaţiile dintre tabele, tot din cadrul paginii principale, prin selectarea legăturii *Delete* deschizându-se o nouă fereastră în care vi se cere să alegeţi relaţiile pe care doriţi să le ştergeţi. De ce spunem relaţii şi nu relaţie? Pentru că în tabela copil pot exista mai mulţi indecşi diferiţi, dar care referă cheia primară din cadrul tabelei părinte. O dată şterse relaţiile, se modifică şi fişierul *RelDF* corespunzător, în cazul în care acesta a fost creat:



Apăsând butonul *Delete*, veţi fi întrebat dacă doriţi sau nu să ştergeţi relaţiile selectate

La final, după ce aţi efectuat toate operaţiile, puteţi să părăsiţi aplicaţia prin selectarea legăturii *User Logout*, în urma căreia sunteţi redirecţionaţi către pagina principală.

6. Comentarii bibliografice

Detalii privind tehnologiile utilizate în cadrul aplicaţiei *DataR* pot fi consultate în cartea *Tehnologii Web*, a lui Sabin Buraga, apărută în 2001 la Editura Matrix Rom. Problematicile – extrem de actuale – referitoare la Web-ul semantic sunt prezentate în volumul S. Buraga, *Semantic Web. Fundamente şi aplicaţii*, Editura Matrix Rom, 2004. Despre limbajul RDF puteţi afla amănunte în specificaţiile şi tutorialele disponibile la *http://www.w3.org/RDF* şi *http://www.w3schools.com/RDF*. De asemenea, se poate consulta şi situl disponibil la adresa *http://www.zvon.org/*. Informaţii referitoare la metadate şi *Dublin Core Metadata Initiative* se găsesc la *http://www.purl.org/*.

În ceea ce priveşte domeniul bazelor de date, mai ales integritatea referenţială, cititorii pot consulta situl *http://www.databasejournal.com/*. De asemenea, recomandăm cartea I. Gilfillan, *Mastering MySQL 4*, Sybex, 2003. O altă adresă care poate prezenta interes este cea a sitului MySQL: *http://www.mysql.com/*.

CAPITOLUL V

Noua provocare în grafica 2D – SVG

Cătălin Bulancea, samprasfan03@yahoo.com

Acest capitol descrie aplicația *RelVis*, un sit Web care realizează vizualizarea unor baze de date relaționale (tabelele și relațiile dintre ele, plus informații despre câmpuri și înregistrări) prin intermediul limbajului *SVG* (*Scalable Vector Graphics*), standard al Consorțiului Web și alternativă de succes la formatul proprietar Macromedia ShockWave Flash. Aplicația *RelVis* este în directă legătură cu aplicația *DataR*, prezentată în capitolul precedent. *RelVis* preia documentele *RelDF* generate de aplicația anterioară și le vizualizează pe Web ca figuri SVG.

1. Introducere

În contextul spațiului WWW, nu mai putem vorbi doar de obținerea de informații necesare utilizatorilor de diferite vârste, ci și de modul de prezentare a acestora. Pentru vizitatorul unui sit, ceea ce contează astăzi este interfața, maniera de afișare, formatul de dispunere a datelor. *Designer*-ul trebuie să realizeze pagini Web interesante, elegante și atrăgătoare pentru utilizator.

O altă componentă esențială este stocarea datelor prin intermediul bazelor de date. Nu există aplicație serioasă care să nu folosească baze de date. Creșterea volumului de informații a dus la necesitatea creării unor baze de date cu un număr imens de tabele și de înregistrări, făcând astfel accesarea din ce în ce mai dificilă și mai greoaie. Un răspuns la această problemă ar putea fi dat de XML, limbaj de marcare care oferă o soluție reală de stocare și interschimbare a datelor.

Aplicația de față, numită *RelVis*, își propune să găsească o rezolvare la cele două probleme expuse anterior – afișarea de informații referitoare la baze de date (tabele, câmpuri, înregistrări) într-o manieră atrăgătoare, prin intermediul unei interfețe elegante, construite în întregime în *SVG* (*Scalable Vector Graphics*), un limbaj care descrie – folosind sintaxa XML – elemente de grafică vectorială 2D.

2. Mai multe despre SVG

SVG este un limbaj care descrie elemente de grafică 2D în XML. Acesta permite utilizarea a trei tipuri de obiecte grafice: forme bazate pe grafică vectorială (căi – *paths* – formate din linii drepte şi curbe), imagini *raster* şi text. Obiectele grafice pot fi grupate, stilizate, transformate. SVG oferă suport pentru transformări încapsulate, decuparea conţinutului (*clipping paths*), măşti de transparenţă (*alpha*), efecte grafice (*filter effects*) şi obiecte de tip şablon (*template*).

Desenele în SVG pot fi interactive şi dinamice. Animaţiile pot fi definite şi redate fie declarativ (incluzând elemente de animaţie în conţinutul SVG), fie prin intermediul programelor script.

Aplicaţii sofisticate ale SVG sunt posibile prin utilizarea limbajelor de *scripting*, care accesează *SVG Document Object Model* (*DOM*), oferind acces complet la toate elementele, atributele şi proprietăţile. Oricărui obiect grafic SVG îi poate fi asociat un set bogat de funcţii de tratare a evenimentelor (*handlers*), precum *onmouseover* şi *onclick*. Datorită compatibilităţii cu alte standarde Web, *scripting*-ul poate fi realizat asupra elementelor XHTML şi SVG în cadrul aceleiaşi pagini.

Documentele SVG pot fi încapsulate în pagini Web. Există diverse *plugin*-uri în Netscape/Mozilla şi Internet Explorer care dau posibilitatea afişării elementelor grafice în SVG. În cele din urmă, în mod sigur, suportul pentru SVG va fi oferit direct de către *browser*, astfel încât să putem include desene SVG în paginile Web depunând un efort la fel de mic ca şi pentru includerea unei imagini GIF sau JPEG într-un document XHTML. Totuşi, importanţa SVG-ului se extinde mult peste domeniul limitat al paginilor Web. SVG va deveni în cele din urmă mediul standard de interschimbare de desene produse pe orice platformă de diverse companii producătoare de software activând în domeniul graficii vectoriale. De altfel, *TinySVG* reprezintă o submulţime SVG destinată realizării de grafică vectorială bidimensională pentru dispozitive *handheld*.

2.1 Explicarea numelui SVG

Scalable

A fi *scalabil* înseamnă a creşte sau a descreşte uniform. În termeni de grafică, scalabil înseamnă a nu fi limitat la o singură dimensiune, fixată, a unui pixel. Pe Web, a fi scalabil înseamnă că o anumită tehnologie poate susţine o funcţionalitate specifică, indiferent de numărul de fişiere, de utilizatori sau de volumul de informaţie vehiculate, pentru o mare varietate de aplicaţii. SVG, fiind o tehnologie grafică pentru Web, oferă această scalabilitate.

O imagine realizată în SVG poate fi micșorată sau mărită. Același document SVG poate fi, în același timp, o diagramă de dimensiuni mici pe un dispozitiv *PalmTop*, de dimensiuni medii în cadrul unui sit Web redat pe un calculator obișnuit sau de dimensiuni foarte mari în cazul proiectării pe un ecran de cinema. De asemenea, o imagine SVG poate fi mărită sau micșorată la rezoluție maximă pe același dispozitiv de afișare. Imaginile SVG nu au dimensiuni absolute și pot fi afișate la diferite rezoluții.

Elementele grafice SVG sunt scalabile pentru că același conținut SVG poate fi de sine stătător (*stand-alone graphic*) sau poate fi referențiat ori inclus în alt conținut SVG, permițând o metodă complexă de ilustrare a unui conținut grafic format din părți.

Vector

Elementele de grafică vectorială conțin obiecte precum segmentele de dreaptă și curbele. Acest lucru conferă o mai mare flexibilitate în comparație cu celelalte formate grafice, de tip *raster* (de exemplu, PNG și JPEG), care trebuie să stocheze informații despre fiecare pixel al contextului grafic. În mod normal, formatele vectoriale pot integra imagini de tip *raster* și le pot combina cu informație vectorială, cum ar fi, de exemplu, existența căilor de decupare a conținutului (*clipping paths*) pentru a realiza ilustrații complexe. SVG nu face excepție, fiind un concurent serios al Flash-ului – vezi și Anexa 1 a lucrării de față.

Graphics

Majoritatea tipurilor de documente XML reprezintă ori informație textuală, ori desemnează date neprocesate. Ele oferă doar posibilități grafice rudimentare, cel mai adesea mai slabe decât elementul `<img>` din HTML. SVG umple acest gol, oferind o descriere bogată, structurată a elementelor de grafică vectorială; poate fi folosit de sine stătător, *stand-alone* (în acest caz, un document SVG este încărcat direct în navigatorul Web; documentul SVG reprezintă pagina Web redată utilizatorului), sau ca un spațiu de nume XML în cadrul altor documente (de exemplu, se pot integra scheme figurate în SVG în cadrul fișierelor MathML folosite la scrierea de formule matematice în XML).

2.2 Tutorial succint de SVG

Definirea unui fragment de document SVG: elementul „`svg`”

Un fragment de document SVG constă într-un număr de elemente SVG conținute în interiorul unui element `svg`. Acest fragment SVG poate fi vid sau poate include de la un singur element grafic SVG, precum `ellipse`, până la o colecție bogată de elemente de tip container (elemente `g`) ori diverse elemente grafice.

Fragmentul SVG poate fi de sine stătător ca un fişier independent, caz în care fragmentul de document SVG este un *document SVG* sau poate fi inclus *inline* ca un fragment într-un document XML părinte (de exemplu, într-o pagină XHTML).

Următorul exemplu ilustrează un conţinut simplu SVG inclus *inline* ca un fragment într-un document XML părinte. De observat folosirea spaţiilor de nume, indicând faptul că elementele `svg` şi `ellipse` aparţin spaţiului de nume SVG:

```
<?xml version="1.0" standalone="yes"?>
<parinte xmlns="http://example.org"
         xmlns:svg="http://www.w3.org/2000/svg">
   <!-- continutul elementului parinte -->
   <svg:svg width="4cm" height="8cm" version="1.1">
      <svg:ellipse cx="2cm" cy="4cm" rx="2cm" ry="1cm"/>
   </svg:svg>
   <!-- ... -->
</parinte>
```

Iată acum un alt exemplu de fragment de document SVG, dar de data aceasta vorbim de un document SVG de sine stătător, care poate fi salvat ca un document independent cu extensia *.svg* :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<svg xmlns="http://www.w3.org/2000/svg"
     width="3.5in" height="1in">
  <title>Primul exemplu de document SVG</title>
  <circle r="30" cx="34" cy="34"
     style="fill: red; stroke: blue; stroke-width: 2" />
</svg>
```

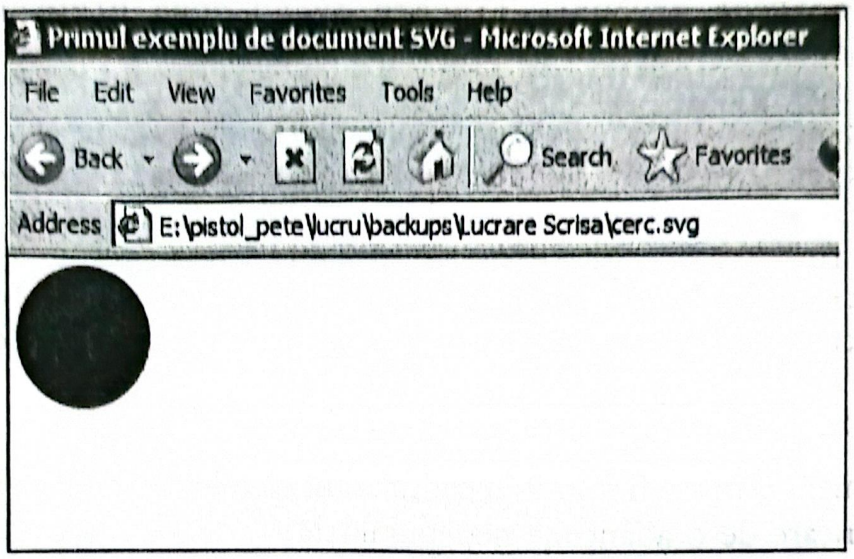
Elementul rădăcină al acestui document SVG este *svg*. Acest element se află în spaţiul de nume *http://www.w3.org/2000/svg*. Uneori, acesta este spaţiul de nume implicit. Alteori, conţine un prefix. Elementul `svg` poate include atributele `width` şi `height`, care specifică dimensiunea „pânzei" (*canvas*) pe care e desenată figura.

Se observă două subelemente: `title` şi `circle`. Elementul `title` conţine un şir de caractere afişat în bara de titlu a *browser*-ului SVG. Elementul *circle* este un element grafic şi reprezintă forma care va fi desenată. Raza cercului desenat este de 30. Se poate pune întrebarea „30 de *ce*: inci, centimetri, pixeli?". Ei bine, 30 înseamnă unităţi în sistemul local de coordonate adimensional. Să ne amintim că SVG este scalabil. Deci dimensiunea de 30 a razei se poate modifica în funcţie de mediul în care se află. Implicit, va fi de 30 de pixeli. Totuşi, se pot realiza transformări care îi vor putea altera dimensiunile (via atributul `transform`).

Atributul `style` desemnează stilul cercului creat, în acest caz, culoarea de fundal, culoarea marginii și grosimea marginii. În loc să folosim atributul `style`, putem utiliza atributele explicite `fill`, `stroke`, `stroke-width`:

```
<circle r="30" cx="34" cy="34"
        fill="red" stroke="blue" stroke-width="2"/>
```

Ca să ne facem o idee despre cum arată desenul tocmai creat, documentul SVG va fi afișat în navigator astfel:



Primul exemplu de document SVG

Elementele `svg` pot apărea în cadrul conținutului SVG (*inner svg*). Acesta este mecanismul prin care fragmente de documente SVG pot fi incluse în alte documente SVG.

Un alt exemplu de utilizare a elementului `svg` este stabilirea unui nou *viewport*.

Majoritatea navigatoarelor nu suportă imagini SVG adăugate direct în HTML. De aceea, se salvează imaginea într-un document SVG separat. Documentul SVG independent creat anterior este inclus într-un document HTML cu ajutorul *tag*-ului `<embed>`. Acesta este foarte asemănător cu elementul `<img>` din HTML. Are atribute ca `width`, `height`, `alt`, `align` și `src`, identice cu cele folosite în HTML. Singura diferență este aceea că elementul `<img>` este folosit pentru formate de imagini suportate implicit de *browser*, în timp ce `<embed>` este folosit pentru formate de imagini care au nevoie de un *plug-in* pentru a putea fi vizualizate. Elementele `<embed>` pot conține atributul `pluginspage`, a cărui valoare este un URL de unde *browser*-ul poate descărca *plug-in*-ul de care are nevoie pentru a afișa conținutul inclus în pagina Web.

```
<embed width="100" height="100" src="circle.svg"
   alt="Un cerc rosu cu margine albastra" align="left"
   pluginspage="http://www.adobe.com/svg/viewer/install/">
```

Gruparea: elementul „g"

Elementul `g` este un element de tip container. Un grup de elemente, precum și diverse obiecte individuale pot primi un nume folosindu-se atributul `id`. Grupurile cu denumire sunt necesare pentru animație și pentru refolosirea obiectelor.

Iată un exemplu de utilizare a grupurilor `g` cu denumire:

```
<svg width="5cm" height="5cm">
  <desc>Doua grupuri, fiecare avand doua dreptunghiuri
  </desc>
  <g id="grup1" fill="red" >
    <rect x="1cm" y="1cm" width="1cm" height="1cm" />
    <rect x="3cm" y="1cm" width="1cm" height="1cm" />
  </g>
  <g id="grup2" fill="blue" >
    <rect x="1cm" y="3cm" width="1cm" height="1cm" />
    <rect x="3cm" y="3cm" width="1cm" height="1cm" />
  </g>
</svg>
```

Un element `g` poate fi folosit în cadrul altui element `g`, deci ar putea avea loc o imbricare de o adâncime nedeterminată.

Reutilizarea obiectelor: elementele „defs" și „use"

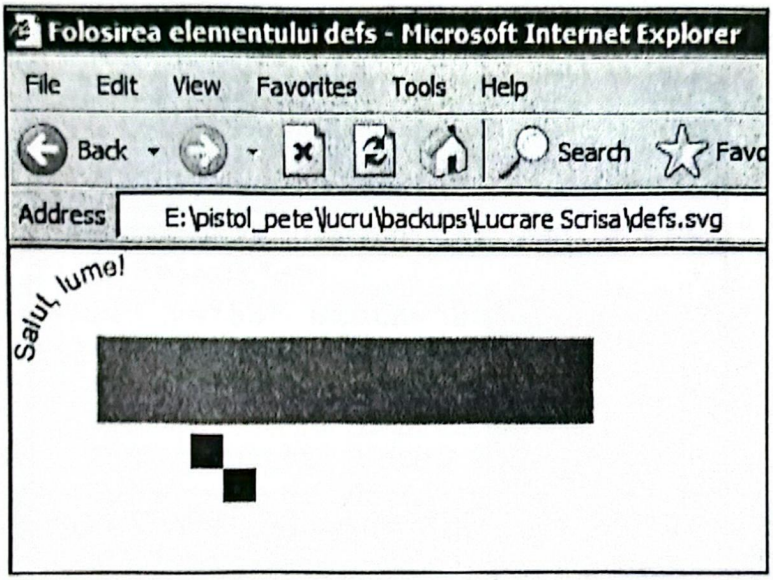
Elementul `defs` este un container pentru elemente referențiate. Pe motiv de reutilizare a obiectelor și pentru o mai ușoară accesibilitate, se recomandă ca elementele referențiate să fie definite în interiorul unui element `defs`.

Elementele copil ale lui `defs` nu sunt desenate direct. Astfel, elementul `defs` seamănă cu elementul `g`, care ar avea setată proprietatea `display` pe `none`. Să presupunem că dorim să folosim un degradé (*gradient*) liniar pentru un element de tip `rect`. În acest caz, elementul care definește gradientul va fi inclus în interiorul *tag*-ului `defs`, la fel și o cale (*path*) pe care vom dori să scriem un text sau un alt element `rect` pe care dorim să îl reutilizăm, schimbându-i doar coordonatele `x` și `y`:

```
<svg width="8cm" height="3cm"
  xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Folosirea elementului defs</title>
  <defs>
    <linearGradient id="Gradient01">
      <stop offset="20%" stop-color="#39F" />
      <stop offset="90%" stop-color="#F3F" />
    </linearGradient>
```

```
    <rect id="rect" width="15" height="15" fill="red"/>
    <path id="textPath" d="M10 50 C10 0 90 0 90 40"/>
  </defs>
  <rect x="1cm" y="1cm" width="6cm" height="1cm"
        fill="url(#Gradient01)"  />
  <use x="80" y="80" xlink:href="#rect"/>
  <use x="95" y="95" xlink:href="#rect"/>
  <text fill="green">
    <textPath xlink:href="#textPath">
       Salut, lume!</textPath>
  </text>
</svg>
```

Iată rezultatul în format SVG:



Folosirea elementului „defs"

Se observă în cod folosirea elementului `use`, care funcționează ca un fel de obiect șablon (*template*) ce poate fi refolosit (instanțiat) în documentul SVG. Elementul `use` referențiază alt element și indică faptul că acesta este desenat la o anume locație dată în document. De aceea, elementul `defs` conține atributele opționale `x`, `y`, `width` și `height`, care sunt folosite pentru a asocia conținutul grafic al elementului referențiat unei anumite zone a sistemului de coordonate curent. Spre deosebire de `image`, elementul `use` nu poate referenția fișiere întregi.

Elementul „image"

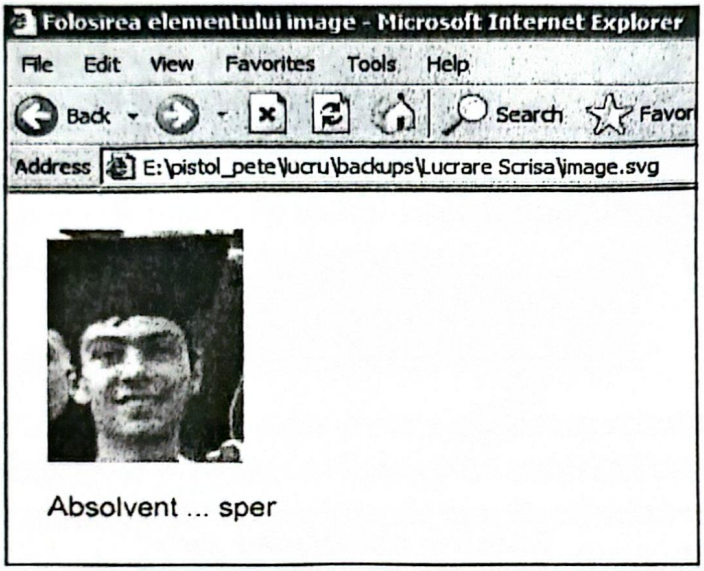
Elementul `image` indică faptul că întreg conținutul unui fișier va fi desenat într-o anumită zonă rectangulară în cadrul sistemului curent de coordonate. Elementul `image` se poate referi la fișiere de format de tip *raster*, cum sunt PNG, JPEG etc.

Un element `image` stabileşte un nou *viewport* pentru fişierul referenţiat. Zonele de demarcare ale noului *viewport* sunt definite de atributele `x`, `y`, `width` şi `height`.

Un exemplu de imagine inclusă într-un fişier SVG este:

```
<svg width="200" height="200"
    xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink">
    <title>Folosirea elementului image</title>
    <image x="20" y="20" width="100" height="130"
           xlink:href="poza_mea.JPG"/>
    <text x="20" y="180" font-size="15">
      Absolvent ... sper</text>
</svg>
```

Şi iată rezultatul includerii imaginii în documentul SVG:



Folosirea elementului „`image`"

Elementul „`rect`"

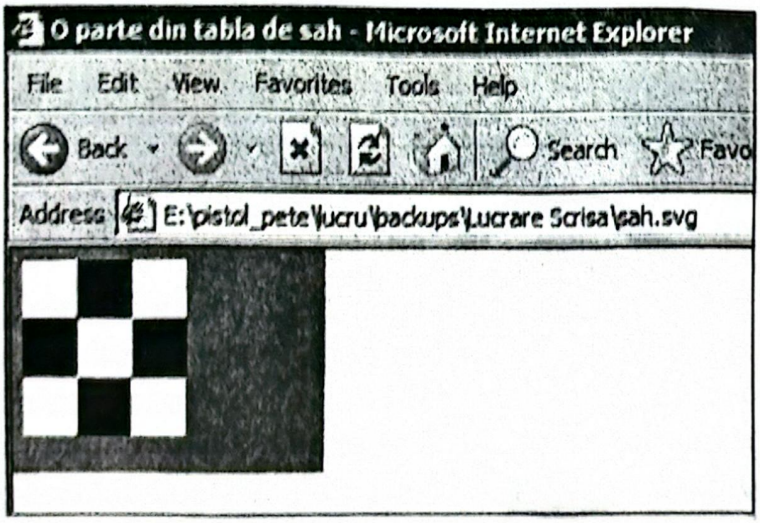
Elementul `rect` reprezintă un dreptunghi aliniat la două axe de coordonate. Fiecare `rect` trebuie să fie complet specificat prin precizarea coordonatelor colţului din stânga sus, a lăţimii şi lungimii. Acestea sunt date de următoarele atribute:

- `x` – coordonata *x* a colţului din stânga sus a dreptunghiului;
- `y` – coordonata *y* a colţului din stânga sus a dreptunghiului;
- `width` – lăţimea dreptunghiului;
- `height` – înălţimea dreptunghiului.

Codul sursă de mai jos generează un fragment al unei tablei de şah, alternând pătratele albe cu cele negre:

```
<svg xmlns="http://www.w3.org/2000/svg"
     width="1.5in" height="1.0in">
  <title>O parte din tabla de sah</title>
  <rect x="0" y="0" width="1.5in" height="1.0in"
        fill="tomato"/>
  <rect x="5" y="5" width="25" height="25"
        fill="floralwhite"/>
  <rect x="30" y="5" width="25" height="25"
        fill="black"/>
  <rect x="55" y="5" width="25" height="25"
        fill="floralwhite"/>
  <rect x="5" y="30" width="25" height="25"
        fill="black"/>
  <rect x="30" y="30" width="25" height="25"
        fill="floralwhite"/>
  <rect x="55" y="30" width="25" height="25"
        fill="black"/>
  <rect x="5" y="55" width="25" height="25"
        fill="floralwhite"/>
  <rect x="30" y="55" width="25" height="25"
        fill="black"/>
  <rect x="54" y="55" width="25" height="25"
        fill="floralwhite" />
</svg>
```

Navigatorul va afișa o imagine precum următoarea:



Folosirea elementului „rect"

Putem realiza dreptunghiuri cu colțuri rotunjite setând atributele `rx` și `ry` ale elementului `rect` la valori pozitive. Cu cât mai mare va fi acest număr, cu atât mai rotunjite vor fi colțurile dreptunghiului.

Elementul „circle"

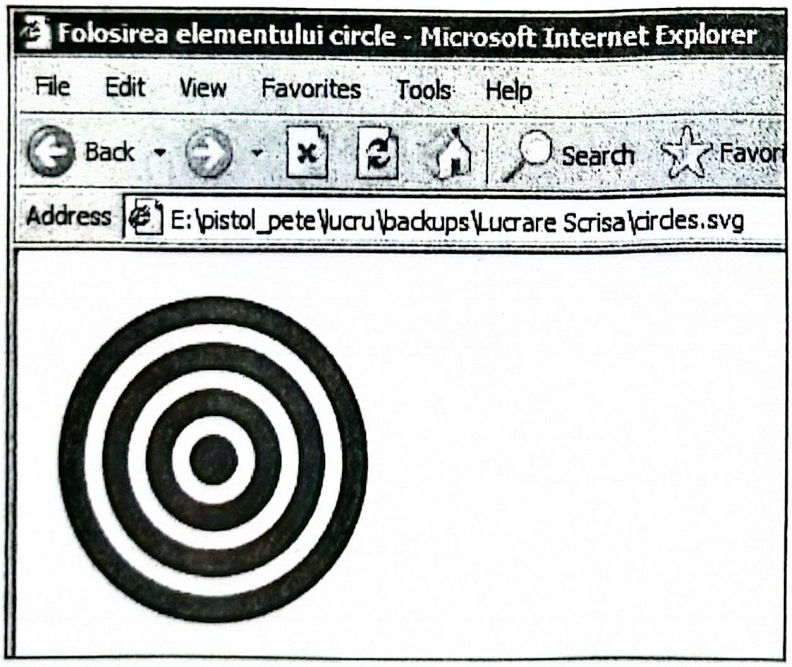
Elementul `circle` reprezintă un cerc. Poziția cercului este determinată de coordonatele centrului său, iar mărimea cercului este dată de raza specificată. Acestea sunt stabilite prin intermediul atributelor:

- `cx` – coordonata *x* a centrului cercului;
- `cy` – coordonata *y* a centrului cercului;
- `r` – mărimea razei.

Figura ce urmează desenează o serie de cercuri suprapuse, așa-numitul *bull's eye*:

```
<svg xmlns="http://www.w3.org/2000/svg"
     width="3.5in" height="2.0in">
<title>Folosirea elementului 'circle'</title>
<circle cx="90" cy="90" r="70"
        fill="red" stroke="black" stroke-width="2" />
<circle cx="90" cy="90" r="60"
        fill="white" stroke="black" stroke-width="2" />
<circle cx="90" cy="90" r="50"
        fill="red" stroke="black" stroke-width="2" />
<circle cx="90" cy="90" r="40"
        fill="white" stroke="black" stroke-width="2" />
```

Iată cum arată „ochiul de taur":



Folosirea elementului „circle"

Elementul „ellipse"

Elipsele au definite axele *x* și *y*. La fel ca dreptunghiurile, elementele de tip `ellipse` au patru atribute care desemnează:

- cx – coordonata *x* a centrului elipsei;
- cy – coordonata *y* a centrului elipsei;
- rx – lungimea razei pe axa *x*;
- ry – lungimea razei pe axa *y*.

De exemplu, această elipsă este de patru ori mai lungă decât lată:

```
<ellipse cx="45" cy="20" rx="40" ry="10" />
```

Elementul „line"

Elementul `line` figurează un segment de dreaptă definit de două puncte. Acesta este identificat de coordonatele *x* și *y* ale acestor puncte, așa cum se specifică în următoarele atribute:

- x1 – coordonata *x* a punctului de start;
- y1 – coordonata *y* a punctului de start;
- x2 – coordonata *x* a punctului final;
- y2 – coordonata *y* a punctului final.

Liniile următoare sunt un exemplu de utilizare a elementului `line`, formând un triunghi:

```
<svg xmlns="http://www.w3.org/2000/svg"
  width="3.5in" height="2.0in">
  <title>Folosirea elementului line</title>
  <line x1="0" y1="100" x2="100" y2="100"
    stroke-width="2px" stroke="blue"/>
  <line x1="0" y1="100" x2="0" y2="0"
    stroke-width="2px" stroke="blue"/>
  <line x1="0" y1="0" x2="100" y2="100"
    stroke-width="2px" stroke="blue"/>
</svg>
```

Elementele „polygon" și „polyline"

Un `polygon` este o formă închisă compusă din linii drepte între o secvență de trei sau mai multe puncte. Primul punct se conectează la al doilea, al doilea la al treilea ș.a.m.d până la ultimul punct, care se conectează la primul. Deci un poligon cu N puncte are N segmente ce unesc punctele. O `polyline` este asemănătoare cu un `polygon`, cu excepția faptului că ultimul punct nu se conectează la primul. Așadar, o formă de tip `polyline` cu N puncte posedă N-1 segmente de dreaptă.

Un exemplu de utilizare a elementelor `polygon` și `polyline` este următorul:

```
<svg xmlns="http://www.w3.org/2000/svg"
     width="3.5in" height="1.5in">
  <title>Folosirea elementelor polygon si polyline</title>
```

```
    <polygon points="0,30 30,0 80,0 110,30 110,80 80,110
       30,110 0,80" fill="white"/>
    <polygon points="120,55 160,40 180,0 200,40 240,55
       200,80 180,120 160,80"/>
    <polyline points="240,30 270,45 312,80 270,110 268,82
       272,23 267,71 311,17 240,30"/>
</svg>
```

Elementul „path"

Alături de aceste şase tipuri de forme de bază, pot fi create o varietate de alte forme prin utilizarea elementului `path`. Elementul `path` reprezintă o curbă arbitrară bidimensională. Căile (`path`-urile) pot fi îngroşate, astfel încât să arate precum liniile, sau pot fi umplute, fiind redate ca obiecte solide. Ne putem gândi la un `path` ca la o curbă desenată de un stilou pe o hârtie. `Path`-urile pot fi conectate sau nu, putând avea stabilite diferite culori şi grosimi.

Există zece operaţii de bază care pot fi efectuate utilizând elementul grafic `path`. Atributul `d` (care provine de la *dată*) conţine instrucţiunile pentru desenarea căii. Aceste instrucţiuni sunt reprezentate de o singură literă:

- `M` şi `m` – pentru mută la... (*move to*);
- `L` şi `l` – pentru trasează o linie la... (*line to*);
- `H` şi `h` – pentru a desena o linie orizontală la... (*horizontal line to*);
- `V` şi `v` – pentru a realiza o linie verticală la... (*vertical line to*);
- `A` şi `a` – pentru a desena un arc de cerc (*elliptical arc to*);
- `C` şi `c` – pentru a genera o curbă Bézier cubică;
- `S` şi `s` – pentru a reda o curbă Bézier *smooth* cubică;
- `Q` şi `q` – pentru a figura o curbă Bézier pătratică;
- `T` şi `t` – pentru a desena o curbă Bézier *smooth* pătratică;
- `Z` şi `z` – pentru a închide calea.

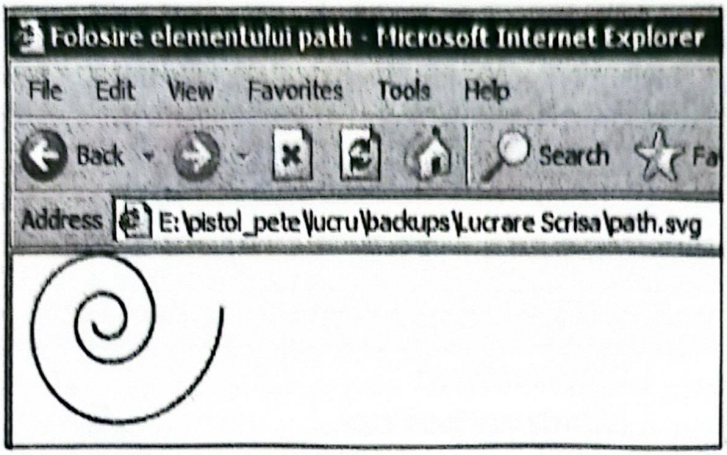
Toate instrucţiunile au două moduri de reprezentare: cu literă mare – coordonate absolute în raport cu sistemul de coordonate – şi cu literă mică – coordonate relative în raport cu sistemul de coordonate curent.

Exemplele de folosire a elementului grafic `path` sunt variate; vom oferi unul în continuare. Iată o spirală realizată cu `path`:

```
<svg>
<title>Folosirea elementului path</title>
<path style="stroke-width: 1; fill-rule: nonzero;
   clip-rule: nonzero; fill: #FFFFFF; stroke: #ff0fca;
   stroke-width: 2; stroke-miterlimit: 4;"
   d="M99.233, 22.5c0, 27.614-22.386, 50-50, 50c-22.091,
      0-40-17.909-40-40c0-17.673, 14.327-32,
      32-32c14.139, 0, 25.6, 11.461, 25.6, 25.6c0,
```

```
        11.311-9.169, 20.48-20.48, 20.48c-9.049,
        0-16.384-7.335-16.384-16.384 c0-7.239,
        5.869-13.107,13.107-13.107c5.791,0,10.486,4.694,
        10.486,10.486c0,4.633-3.756,8.389-8.389,
        8.389c-3.707,0-6.711-3.005-6.711-6.711"/>
</svg>
```

În exemplul anterior s-a folosit instrucțiunea „c", deci vom desena o curbă Bézier cubică:



Utilizarea elementului „path"

Elementul „text"

În SVG, afișarea textului se realizează prin intermediul elementului text, care va reda conținutul său la coordonatele specificate de utilizator. Textul poate aparține unei imagini, poate fi așezat la o anumită poziție sau poate urma o anumită cale (path). Putem alege stilul textului prin folosirea atributului `style` setând `font-family`, `font-size`, `font-style`, `fill`, `stroke`. O altă trăsătură importantă este posibilitatea folosirii caracterelor non-latine, dacă avem instalate fonturile corespunzătoare. Obligatoriu, elementul `path` trebuie să conțină următoarele atribute:

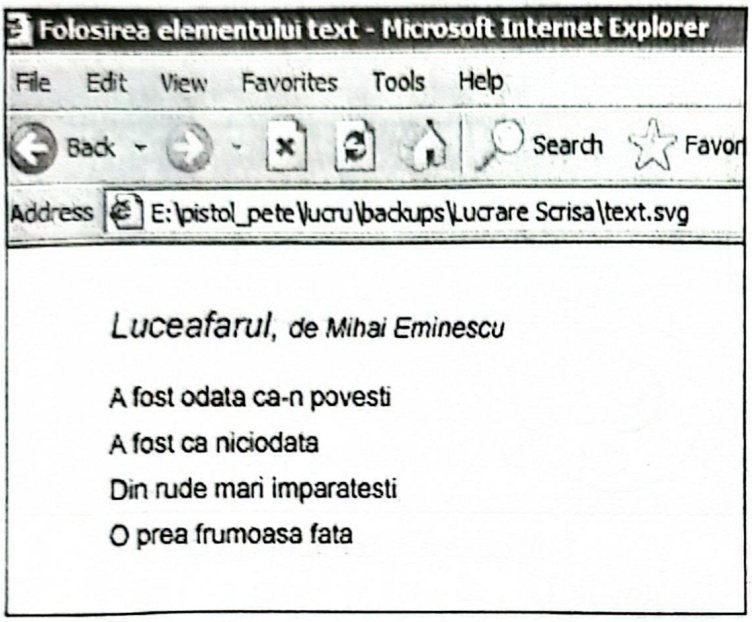
- x – coordonata *x* a textului;
- y – coordonata *y* a textului.

Poemul *Luceafărul* de Mihai Eminescu poate avea în SVG următoarea formă:

```
<svg>
  <title>Folosirea elementului text</title>
  <text x="50" y="20">A fost odata ca-n povesti</text>
  <text x="50" y="40">A fost ca niciodata</text>
  <text x="50" y="60">Din rude mari imparatesti</text>
  <text x="50" y="80">O prea frumoasa fata</text>
</svg>
```

Nu putem integra conținutul poeziei într-un singur element `text`, pentru că singurul lucru pe care SVG nu îl poate realiza este păstrarea spațiilor între versuri. Aceasta se datorează faptului că multe limbi, precum cea chineză, arabică sau tibetană, au reguli complicate de spațiere între cuvinte și linii.

Figura următoare ne înfățișează prima strofă din *Luceafărul* în navigatorul Web:



Folosirea elementului „`text`"

Transformări

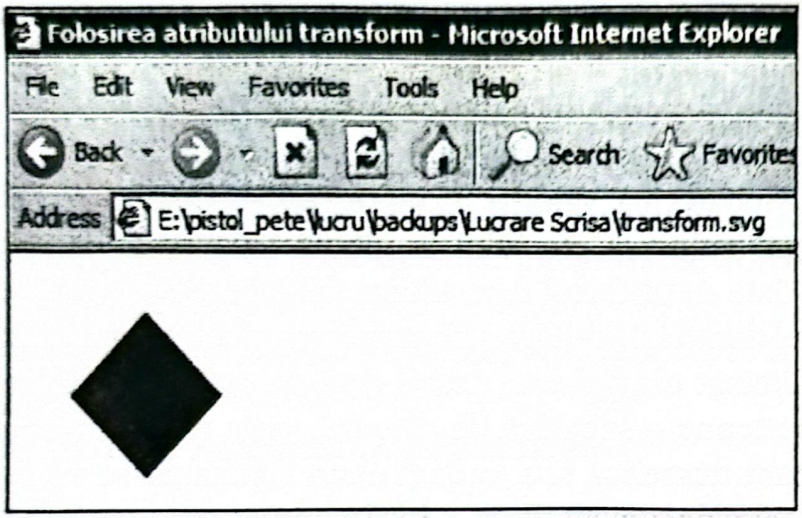
Atributul `transform` este cel care se ocupă cu transformări asupra elementelor SVG. Valoarea atributului `transform` primește o listă de definiții de transformare, care sunt aplicate în ordinea în care sunt scrise. Definițiile sunt separate unele de altele prin spațiu sau virgulă. Tipurile existente de definiții de transformări sunt:

- `matrix (<a> <b> <c> <d> <e> <f>)` – specifică o transformare de forma unei matrici de transformare de forma [a b c d e f];
- `translate (<tx> <ty>)` – desemnează o translație pe axa *x* și/sau *y*;
- `scale (<sx> <sy>)` – specifică o scalare pe axa *x* și/sau axa *y*;
- `rotate (<unghi> [<cx> <cy>])` – specifică o rotație cu unghiul specificat în jurul punctului dat de coordonatele *cx* și *cy*;
- `skewX (<unghi>)` – desemnează o transformare de alungire pe axa *x* cu unghiul dat ca parametru;
- `skewY (<unghi>)` – specifică o transformare de alungire pe axa *y* cu unghiul dat ca parametru.

După cum am menționat, ordinea în care se aplică transformările este dată de cea în care sunt scrise în atributul `transform`, ca în exemplul:

```
<svg>
  <title>Folosirea atributului transform</title>
  <g transform="translate(-10,-20) scale(0.5)
       rotate(45 100 100) translate(5,10)">
     <rect x="120" y="50" width="100" height="100"
          fill="green"/>
  </g>
</svg>
```

Această transformare va transla mai întâi dreptunghiul pe axa x cu –10 şi pe y cu –20, apoi îl va scala de 2 ori şi în cele din urmă îl va roti cu 45 de grade:



Folosirea atributului „transform"

Animaţii

Animaţiile reprezintă poate partea cea mai interesantă a SVG-ului, fiind realizate cu ajutorul următoarelor elemente:

- `animate` – folosit pentru a anima în timp o singură proprietate sau atribut;
- `set` – utilizat doar pentru a seta la anumite valori un atribut; este neaditiv;
- `animateMotion` – determină un obiect anume să se mişte de-a lungul unei căi trasate;
- `animateColor` – specifică o transformare în timp a culorii unui obiect;
- `animateTransform` – animă o transformare conform tipului transformării; atributul `type` poate lua valorile: `translate`, `scale`, `rotate`, `skewX`, `skewY`, permiţând astfel controlul translaţiei, scalării, rotaţiei şi/sau schimbării unghiului de vizualizare.

Aceste elemente conţin atribute ca:

- `from` – specifică valoarea de start al animaţiei;
- `to` – stabileşte valoarea de final al animaţiei;
- `by` – desemnează o valoare relativă între valorile specificate de `from` şi `to`;

- `dur` – specifică durata animației în timp;
- `begin` – referă momentul de start al animației, în urma cărei acțiuni va porni animația; poate fi, de exemplu, `buton.click`, ceea ce înseamnă că la apăsarea butonului va porni animația, sau `2s`, însemnând că animația va porni la două secunde de la încărcarea documentului SVG;
- `end` – specifică momentul de sfârșit al animației, acțiunea efectuată pentru a opri animația; poate fi, de exemplu, `buton.mouseout`, ceea ce înseamnă că, în momentul în care pointerul mouse-ului iese din cadrul butonului, animația se va opri;
- `additive` – verifică dacă animația este sau nu aditivă; poate lua valorile `sum` (specifică faptul că animația se va adăuga unei animații precedente cu nivel de prioritate mai scăzut) și `replace` (animația va suprascrie o animație anterioară cu nivel de prioritate mai scăzut; aceasta este valoarea implicită);
- `fill` – poate lua valorile `freeze` (acest efect este definit pentru a „îngheța" valoarea efectului la ultima valoare de la sfârșitul duratei active date de atributul `dur`; ultima valoare rămâne „înghețată" până la sfârșitul duratei de redare a documentului) și `remove` (efectul animației este eliminat când durata activă dată de atributul `dur` se termină);
- `repeatCount` – specifică de câte ori se va repeta animația; poate lua o valoare numerică sau `indefinite` (animația se va repeta până la închiderea documentului).

Elementele asupra cărora se pot aplica animațiile, definite sunt: `g`, `defs`, `use`, `image`, `switch`, `path`, `rect`, `circle`, `ellipse`, `line`, `polyline`, `polygon`, `text`, `clipPath`, `mask`, `a` și `foreignObject`.

Posibilitățile de animație oferite de SVG sunt, de fapt, aceleași cu cele ale limbajului *SMIL* (*Synchronized Multimedia Integration Language*).

Iată un exemplu de folosire a unei animații asupra unui pătrat:

```
<rect x="120" y="50" width="100" height="100"
      fill="green"  opacity="1">
   <animate attributeName="opacity" from="1" to="0"
      begin="mouseover" dur="2s"
      repeatCount="indefinite"/>
</rect>
```

Animația se va executa de un număr nedefinit de ori și va transforma gradul de opacitate a pătratului de la valoarea 1 la valoarea 0, într-un interval de `2s`, atât timp cât cursorul mouse-ului se află deasupra sa.

3. Descrierea aplicației *RelVis*

3.1 Scopul aplicației. Specificarea cerințelor

Pentru a determina scopul aplicației care face subiectul acestui capitol trebuie să facem o introducere asupra datelor de intrare necesare. Acestea sunt furnizate de *DataR*, aplicația detaliată în capitolul precedent. Reamintim cititorului că *DataR* realizează modelarea relațiilor dintre tabelele unor baze de date și reprezentarea lor folosind *RDF* (*Resource Description Framework*). Reprezentările RDF generate sunt denumite în continuare *RelDF*.

RelVis preia documentele *RelDF* și le vizualizează ca figuri SVG. *RelVis* își propune să construiască în format SVG o aplicație Web simplu de utilizat, eficientă, flexibilă, capabilă să realizeze interacțiunea dinamică cu utilizatorul. Vom avea deci reprezentate mai multe baze de date, tabelele fiecărei baze de date în parte, relațiile dintre aceste tabele, câmpurile fiecărei tabele, iar pentru fiecare câmp vom afișa dinamic înregistrările sortate în ordine crescătoare.

Tehnologiile folosite de aplicație vor fi PHP, MySQL, RDF, SVG și JavaScript.

3.2 Modularizarea. Analiza și proiectarea

Se impune realizarea unei modularizări a aplicației în vederea unei ușoare gestiuni a codului sursă. Distingem trei module importante care compun aplicația:

- *Modulul PHP* – în care se descriu algoritmii și metodele PHP care generează dinamic documentul SVG;
- *Modulul SVG* – în care descriem obiectele create (desenate) în SVG: ferestre de dialog, butoane, controale, meniuri etc.
- *Modulul JavaScript* – în care definim interacțiunea utilizatorului cu aplicația prin intermediul limbajului JavaScript.

Modulul PHP

Să detaliem primul modul, realizat în PHP. Fișierul *RelVis.php* constituie piatra de temelie a aplicației. Aici construim dinamic documentul SVG cu ajutorul funcției `fwrite(resource handle, string string [, int length])`. Funcția `fwrite()` scrie conținutul șirului `string` în fișierul către care pointează `handle`.

Mai întâi creăm un fișier în format SVG, numit *RelVis.svg*, cu ajutorul funcției `fopen(string filename, string mode)`. Valoarea returnată de funcție este un descriptor de fișier, `$file`, folosit ca prim parametru în cadrul lui `fwrite()`. Parametrul `filename` este numele fișierului care se dorește a fi deschis. În cazul nostru, apelul `fopen()` va crea fișierul *RelVis.svg*, deoarece acesta nu există. Desigur, în cazul în care nu se reușește crearea fișierului, se va afișa un mesaj de eroare.

O dată creat descriptorul de fişier, vom putea scrie în fişierul *RelVis.svg* utilizând funcţia `fwrite()` descrisă mai sus. Deoarece SVG provine din XML, prima linie a documentului va fi declaraţia unui document XML obişnuit:

```
fwrite($file,
  '<?xml version="1.0" encoding="iso-8859-1"?>');
```

Următoarele linii de genul `fwrite(...)` vor construi întreg documentul SVG într-o manieră dinamică. Aceasta înseamnă că vom încărca într-un arbore *DOM* (*Document Object Model*) conţinutul documentului *RelDF* creat de aplicaţia *DataR* şi, în funcţie de nodurile întâlnite, vom crea (desena) tabele, câmpuri, relaţii între tabele etc.

Documentul de care avem nevoie (datele reprezentând „materia primă"), *RelDF*, se generează şi el dinamic printr-un script PHP. Aşadar, ca să avem acces la el, vom rula scriptul cu parametrii de care are acesta nevoie şi îl vom stoca într-o variabilă `$RDFfile`, din care vom încărca DOM-ul:

```
$RDFfile = "http://localhost/sib/fullRDF.php?"
  "h=$host&amp;u=$user&amp;p=$password";
...
if (!$dom = domxml_open_file($RDFfile)) {
    echo "Eroare de procesare a documentului.\n";
    exit;
}
```

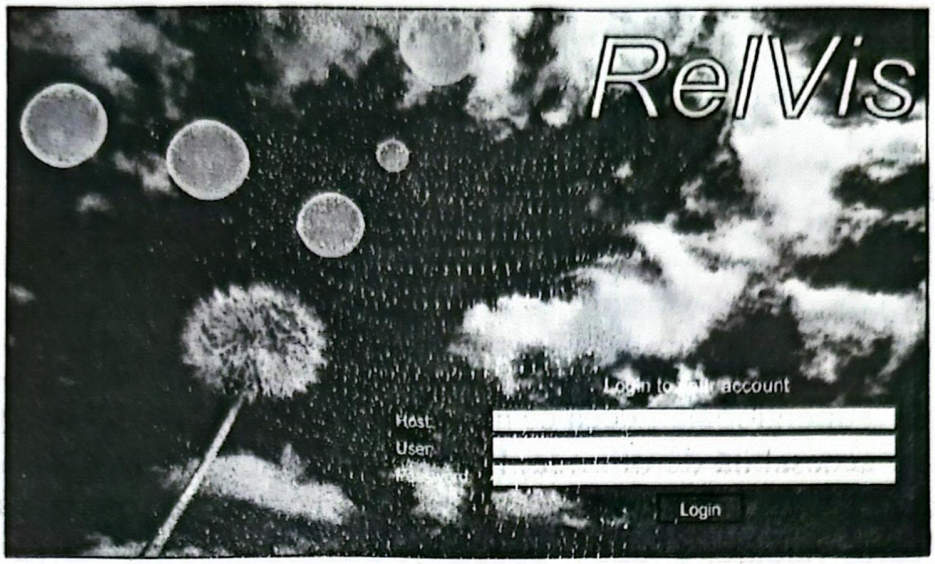
Parametrii `h`, `u` şi `p` desemnează gazda, utilizatorul, respectiv parola şi sunt necesari pentru că vom utiliza o bază de date MySQL în care accesul e restricţionat în funcţie de statutul utilizatorului, administrator sau nu.

Scriptul *fullRDF.php* creează documentul *RelDF*. Se observă că valoarea atribuită variabilei `$RDFfile` este un URL, deci conţinutul variabilei va fi cel al documentului aflat la adresa respectivă.

Variabila `$dom` va stoca o structură arborescentă formată din conţinutul lui `$RDFfile`, procesată apoi de script pentru determinarea datelor de intrare necesare algoritmului aplicaţiei.

În vederea optimizării vitezei de execuţie a aplicaţiei, nu se va executa decât o singură dată scriptul *fullRDF.php*, deoarece arborele DOM din `$dom` va fi salvat în forma unui document cu extensia *.rdf* pe server. Astfel, a doua oară când vom încărca DOM-ul, acesta va fi preluat direct din fişierul local aflat pe disc, îmbunătăţindu-se astfel performanţele programului.

Mai jos, vom descrie autentificarea unui utilizator în contul său prin fereastra de dialog *Login*. Este necesară doar introducerea gazdei (*host*), care poate fi *localhost* sau o adresă dată de utilizator. În cazul furnizării valorii *localhost*, completarea câmpurilor *user* şi *password* este opţională (în această situaţie, vizitatorul va fi considerat ca fiind anonim – *Anonymous*).



Fereastra *Login* pentru accesul utilizatorului în contul său

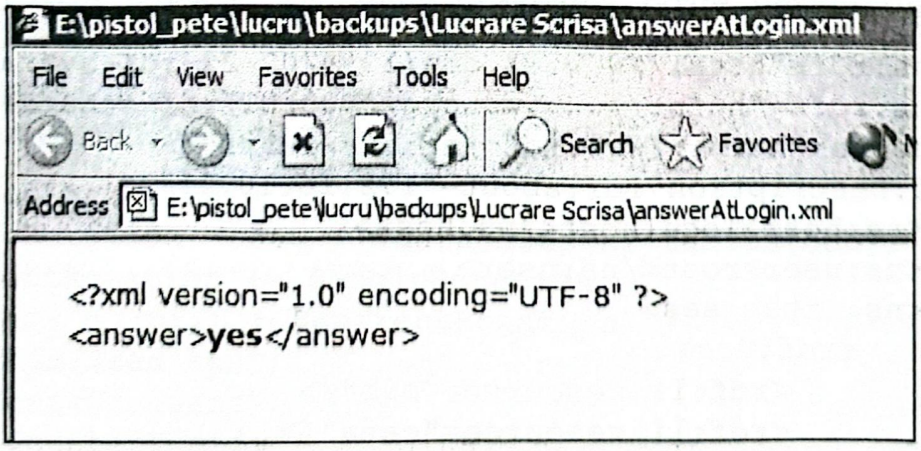
După cum se poate observa, accesul la aplicație este securizat, deoarece accesul la bazele de date din MySQL se realizează pe bază de conturi de utilizatori. Astfel, administratorul (*root*) va avea cele mai multe drepturi, iar *Anonymous* – cele mai puține.

La apăsarea butonului *Login* rulează un script PHP:

```
answerAtLogin.php?h=" + phrase[0] + "&u=" +
phrase[1] +  "&p=" + phrase[2]
```

care verifică validitatea gazdei, a numelui de utilizator și a parolei acestuia. Programului i se transmit parametrii `h`, `u` și `p`. La rularea sa, se va genera automat un fișier XML care va conține răspunsul la încercarea de a intra, în funcție de corectitudinea datelor introduse.

Documentul XML are structura din figură:



Răspunsul la intenția de intrare în cont a utilizatorului

URL-ul asociat execuției scriptului ce are rezultatul de mai sus este: *http://localhost/licenta/RelVis.php*. Se observă că nu este setat parametrul `login`, adică vom intra mai întâi în fereastra de autentificare. Această acțiune este realizată de următorul cod simplu:

```
if(isset($_GET['login']))
{
     $login = $_GET['login'];
}
else
{
     $login=1;
}
```

După cum cititorul probabil deja cunoaşte, variabila `$_GET` este o variabilă predefinită „superglobală", ceea ce înseamnă că este vizibilă în întreg scriptul PHP şi este accesabilă de oriunde, fiind folosită pentru a transmite variabile către script cu ajutorul parametrilor. În cazul nostru, se trimite parametrul `login` a cărui valoare, `$_GET['login']`, e atribuită variabilei `$login`.

Parametrul `login`, nefiind setat, va determina ca variabila `$login` să fie iniţializată cu valoarea 1. Aceasta înseamnă că se va apela funcţia de desenare a ferestrei de conectare.

```
if($login == 1)
{
    LoginDialog();
}
```

Unul dintre algoritmii importanţi ai aplicaţiei este cel de dispunere a bazelor de date în zona client. În urma citirii şi procesării documentului RDF, se va determina numărul de baze de date conţinute în acesta.

Vom prezenta o scurtă porţiune a documentului RDF care conţine informaţii privitoare la bazele de date:

```
<?xml version="1.0" encoding="UTF-8" ?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:ns="http://www.infoiasi.ro/reldf">
  <rdf:Description rdf:about="172.16.10.117">
      <ns:host>localhost</ns:host>
      <ns:user>root</ns:user>
      <ns:databases>
          <rdf:Bag>
              <rdf:li resource="sib"/>
              <rdf:li resource="cata"/>
              <rdf:li resource="icy"/>
              ...
          </rdf:Bag>
      </ns:databases>
  </rdf:Bag>
</rdf:RDF>
```

Porţiunea de cod utilizată pentru determinarea numelui fiecărei baze de date este următoarea:

```
$root=$dom->document_element();
//determinam numele bazelor de date si
//le stocam in databases_names
$databases_names = array();
$tag_db = $root->get_elements_by_tagname("databases");
$li = $tag_db[0]->get_elements_by_tagname("li");
foreach($li as $val) {
 array_push($databases_names,
   $val->get_attribute("resource"));
}
//determinam numarul de baze de date
$nr_db = count($databases_names);
```

Determinăm elementul <ns:databases> ce conţine un container <rdf:Bag> incluzând la rândul său elemente <rdf:li> care au ca atribute resource numele bazei de date. Vom reţine aceste nume în tabloul $databases_names, iar numărul lor este dat de funcţia count($databases_names).

Acestea trebuie reprezentate grafic sau, mai bine zis, dispuse în zona client în aşa fel încât să fie afişate în întregime. Pentru aceasta, am conceput un algoritm de calculare a coordonatelor fiecărei baze de date. Zona de afişare (zona client) nu poate fi depăşită în lăţime, deoarece am folosit un număr fix de baze de date. Dacă zona client este depăşită în înălţime, atunci vom face vizibil un *scroll* vertical, având astfel posibilitatea de a vizualiza toate bazele de date.

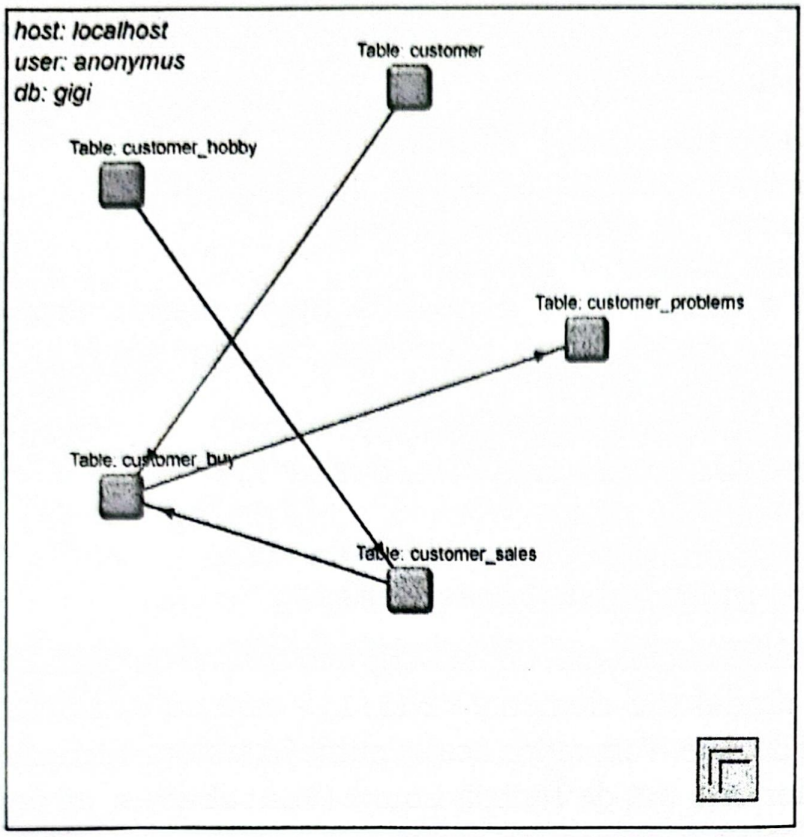
Un algoritm şi mai interesant este cel în care pentru fiecare bază de date în parte afişăm tabelele. Deoarece desenarea poate deveni dificilă în condiţiile în care avem multe tabele, am ales o dispunere oarecum logică, în cerc, a acestora.

Codul utilizat în acest algoritm este următorul:

```
$currentRotation = 0;
$angleRotation = 360 / $databases_nr_tables[$db];
$currentRotation += $angleRotation;
$radians = (pi() * $currentRotation) / 180;
$y = sin($radians);
$x = cos($radians);
$y = $centerY + $y * $radius;
$x = $centerX + $x * $radius;
```

unde cercul este dat de punctele ce constituie tabelele: unghiul cu care rotim o tabelă este $angleRotation, rotirea curentă este $currentRotation, iar coordonatele tabelei sunt $x şi $y.

Să vedem acum maniera de vizualizare construită prin algoritm:



Reprezentare pe un cerc a tabelelor unei baze de date

Se observă cum în baza de date folosită ca exemplu sunt reprezentate tabelele *customer_sales*, *customer_buy*, *customer_hobby*, *customer* şi *customer_problems*, precum şi relaţiile dintre acestea. Se mai remarcă, în partea din dreapta jos, butonul de minimizare, care readuce utilizatorul în fereastra de vizualizare a tuturor bazelor de date.

Această reprezentare este totuşi una temporară, deoarece acţiunea cu mouse-ul de *drag and drop* asupra unei tabele duce la mutarea acesteia în zona dorită de utilizator.

Desenarea tabelelor şi a legăturilor dintre ele se realizează în funcţiile `drawTables($db)` şi `drawPaths($db)`, care primesc ca parametri bazele de date respective.

Am discutat anterior despre parametrii necesari aplicaţiei, trimişi prin URL. Dintre aceştia, doi sunt foarte importanţi: `$open` şi `$save`. Ei ne indică dacă la execuţia scriptului PHP se va desena fereastra de dialog pentru salvarea unui fişier sau pentru deschiderea lui. Acest lucru se realizează prin funcţia `$FileDialog(n)`, unde `n` are valoarea 0 sau 1, în funcţie de ce fereastră de dialog doreşte să deschidă utilizatorul.

Aici folosim o funcţie de citire din director a tuturor fişierelor conţinute în acesta. Vom face distincţie între fişierele cu extensia *.rdf* şi celelalte, deoarece dorim să afişăm, pe rând, numai fişierele *.rdf* sau toate fişierele din directorul *./saved*, în funcţie de opţiunea aleasă în *combo-box*-ul ferestrei de dialog. Iată cum se realizează acest lucru în PHP:

```
$all_files = array();
$rdf_files = array();
if($handle = opendir('./saved')) {
  while(false !== ($f = readdir($handle))) {
    if($f != "." && $f != "..") {
      if(strpos($f, ".rdf") !== false) { // am gasit
         array_push($rdf_files, $f);
      }
      array_push($all_files, $f);
    }
  }
}
```

Întâi se face deschiderea directorului amintit prin `opendir('./saved')` și apoi, cât timp avem ce citi din acesta, `readdir($handle)`, stocăm în tabloul `$rdf_files` doar numele fișierelor cu extensia *.rdf*, apoi în tabloul `$all_files` numele tuturor fișierelor, indiferent de extensie, inclusiv pe cele cu extensia *.rdf*.

O altă metodă PHP care se ocupă de lucrul cu fișiere este `copy(string sursa, string destinatie)`. Aceasta este folosită atunci când se dorește salvarea pe disc a unui fișier peste unul care deja există, deci o suprascriere.

Iată cum se realizează suprascrierea:

```
<?php
if(!copy("./saved/".$old_file, "./saved/".$RDFfile)) {
    ?>
    <script type="text/javascript">
    alert("Failed to copy <? echo $old_file;?>
       in  ./saved/<? echo $RDFfile;?>");
    </script>
    <?php
}
?>
```

unde `$old_file` este fișierul sursă, care va realiza o copie a sa peste `$RDFfile`, destinația.

Meniul aplicației reprezintă unealta de bază pentru utilizator. Opțiunile acestuia sunt:

- *Document* – conține operațiile *Open* și *Save* prin care se pot încărca sau salva pe disc fișiere *.rdf*;
- *Databases* – include lista de baze de date care pot fi accesate prin maximizare și minimizare;
- *Help* – oferă opțiunile *Contents* și *About RelVis*, prin care se poate obține ajutor în manipularea aplicației și vedea fereastra *About* cu informații despre versiunea programului și autor.

Desenarea meniului aduce cu sine o problemă: numărul bazelor de date din submeniul *Databases* poate depăşi înălţimea zonei alocate pentru meniu. Există două soluţii la această problemă:

- implementarea unui *scroll* vertical;
- implementarea de controale de tip *tab* de organizare a datelor.

Prima variantă are totuşi dezavantajul unei accesări mai greoaie şi a pierderii din raza vizuală a submeniului *Document* o dată cu apăsarea *scroll*-ului în jos. De aceea am optat pentru a doua variantă, mai rapidă şi mai estetică. *Tab*-urile vor fi desenate doar dacă numărul lor depăşeşte valoarea variabilei `$max_visible_db`. Astfel, avem o condiţie pusă în cod care ne spune să desenăm sau nu *tab*-urile:

```
if($nr_db > $max_visible_db) {
  //deseneaza tab-urile
}
```

Nu se ştie dacă numărul bazelor de date va depăşi sau nu numărul total de baze de date vizibile în zona meniului, aşa că vom avea o parcurgere iterativă de genul:

```
for($i = 0; $i < (($nr_db > $max_visible_db) ?
                  $max_visible_db : $nr_db); $i++) {
//deseneaza bazele de date
}
```

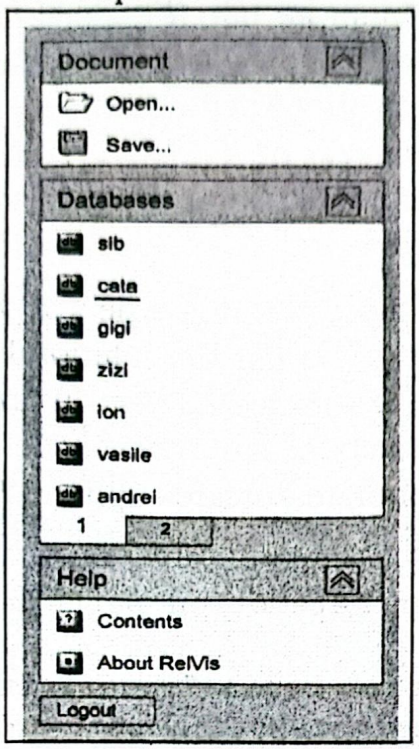
Modulul SVG

Este partea cea mai spectaculoasă a aplicaţiei, introducându-ne în universul SVG şi oferind, într-o mică măsură, o privire de ansamblu asupra posibilităţilor sale. În acest modul vom construi în întregime interfaţa grafică a aplicaţiei. Totul este desenat prin folosirea formelor elementare `rect`, `circle`, `line`, `text`, precum şi a elementelor mai sofisticate, cum sunt `path` sau degradeurile şi filtrele.

Trebuie să realizăm o structurare bine definită a părţii de grafică, pentru că interfaţa cu utilizatorul este partea cea mai importantă a aplicaţiei. Interfaţa se doreşte a fi una atrăgătoare, prietenoasă, uşor de folosit, rapidă şi eficientă în acelaşi timp. De aceea, am mizat pe construirea unei interfeţe atât pentru utilizatorul care are cunoştinţe medii de folosire a calculatorului, cât şi pentru cel avansat în domeniu.

Culorile sunt calde, controalele mari şi uşor de utilizat, iar metaforele lor sunt uşor de recunoscut. Astfel, avem în faţă o aplicaţie grafică constituită dintr-un meniu şi o zonă client, în care sunt figurate bazele de date cu care putem interacţiona.

SVG este un limbaj de marcare foarte flexibil şi cu capacităţi remarcabile de realizare a unei grafici capabile să concureze cu deja faimosul Flash. SVG aduce însă ceva în plus: posibilitatea realizării de animaţii doar prin intermediul elementelor SVG, deci fără necesitatea unui limbaj de programare, a unor linii de cod suplimentare.



Vizualizarea meniului aplicaţiei

Vom descrie elementele utilizate în desenarea meniului aplicaţiei, precum şi animaţiile pe care le efectuează acestea (vezi figura de mai sus).

Pentru realizarea sa am creat câte un grup `<g>` pentru fiecare dintre elementele meniului. Astfel, avem grupul cu identificatorul `group_document` pentru elementul corespunzător *Document*, `group_databases` pentru *Databases*, `group_help` pentru *Help* şi `group_logout` pentru opţiunea *Logout*.

În acest mod, ne putem referi la ele atunci când efectuăm operaţii asupra lor. Cele mai importante operaţii sunt cele de expandare şi colapsare (*expand/collapse*), cele de interacţiune cu subopţiunile meniului, precum şi folosirea *tab*-urilor.

Să realizăm un succint studiu de caz: dorim să deschidem elementul de meniu *Document*. Pentru a reuşi să translatăm opţiunile de meniu de dedesubt, va trebui să declarăm grupurile opţiunilor imbricate, unul în altul. Deci grupul părinte va fi `group_document`, iar în interiorul său includem grupul bazelor de date, care vor stoca la rândul lor grupul pentru *Help* ş.a.m.d:

```
<g id="group_document">
   ...
   <g id="group_ databases">
```

```
    ...
    <g id="group_help">
      ...
        <g id="group_logout">
        ...
        </g>
      ...
    </g>
    ...
  </g>
  ...
</g>
```

Declarându-le în acest mod, o translare pe axa *y* a opțiunii *Document* va duce la translarea tuturor grupurilor din interiorul său. Va trebui calculată totuși mărimea translării pe axa *y* a fiecărui *item*. În cazul opțiunii *Document*, nu este necesar calculul, în schimb, pentru toate cele de dedesubtul acestuia, da.

Iată animația pentru coborârea/urcarea opțiunii *Help*, realizată cu ajutorul unui element de animație în SVG, `animateMotion`:

```
fwrite($file, '<animateMotion path="m 0 0 v 0 ' .
($db_h)
  . '" fill="freeze" begin="indefinite" dur="1s"/>');
fwrite($file, '<animateMotion path="m 0 ' . ($db_h)
  . ' v 0 '.(-$db_h)
  . '" fill="freeze" begin="indefinite" dur="1s"/>');
```

Atributul `begin` indică momentul începerii animației. Aici, el ia valoarea `indefinite`, semnificând că va trebui să pornim noi înșine animația. Și o facem atunci când apăsăm pe butonul corespunzător opțiunii de meniu *Databases*. Atributul `begin` ar fi putut lua, de exemplu, valoarea `3s`, ceea ce înseamnă că animația va porni la 3 secunde după încărcarea documentului, sau valoarea `databases.click`, însemnând că la apăsarea butonului *Databases* va porni animația. Dacă însă am fi folosit această variantă sau prima, nu am fi știut dacă atunci când apăsăm avem de-a face cu o operație de expandare sau de restrângere.

Astfel, stabilim valoarea `indefinite` pentru atributul `begin` și vom adăuga un atribut propriu, numit `op`, adică operația care urmează să fie efectuată. Deoarece la început toate meniurile sunt restrânse, toate atributele `op` vor avea valoarea `expand`, adică urmează să inițiem o operație de expandare:

```
fwrite($file, '<rect id="databases" op="expand"
  x="0" y="0" width="' . ($clientX-40) .
  '" height="'   $item_height . '" opacity="0"
  onclick="ExpandCollapse(evt)"/>');
```

La apăsarea butonului, se va apela metoda JavaScript `ExpandCollapse(evt)`, care va porni animația în funcție de valoarea atributului `op`.

Să trecem la desenarea zonei client, cea în care se afișează efectiv bazele de date, tabelele din acestea, câmpurile și înregistrările. Am discutat anterior despre maniera de dispunere a bazelor de date în fereastra client. Am văzut că avem un număr fix în lățime (3), dar un număr variabil în înălțime, deci avem nevoie de implementarea unui *scroll* vertical.

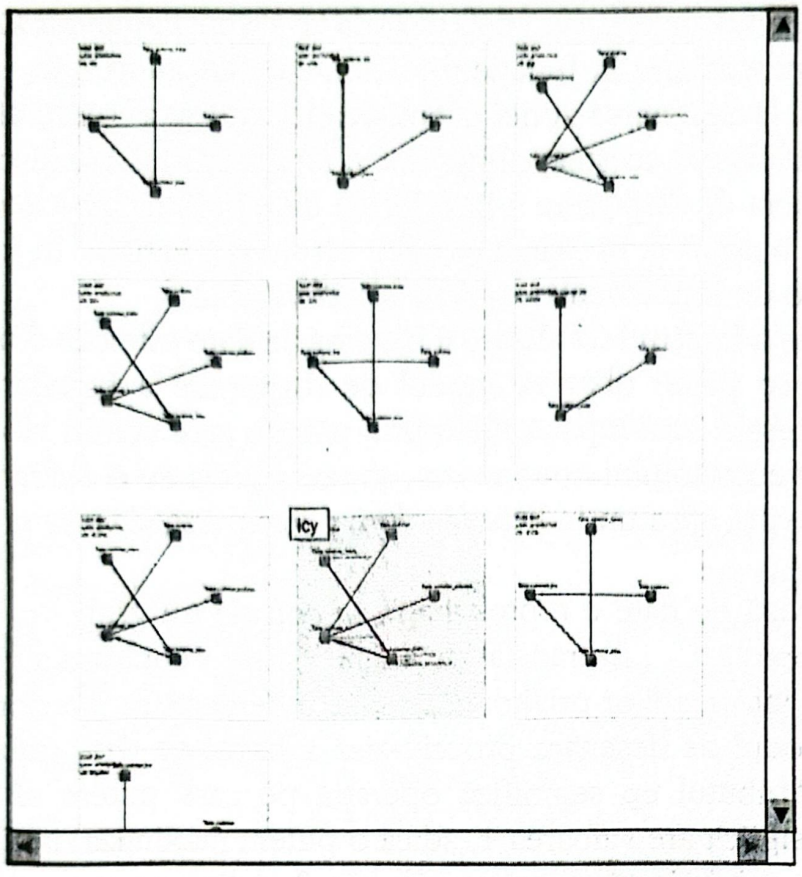
O imagine a ferestrei de afișare a bazelor de date este cea din următoarea figură. Aici se poate observa modul de dispunere a bazelor de date în fereastră. Acestea sunt reprezentate prin pătrate care conțin tabele și relații; la apariția evenimentului `mouseover`, se vor selecta cu o culoare diferită de a restului și vom afișa un indiciu (*tooltip*) pentru vizualizarea numelui bazei de date.

Fiecare bază de date e reprezentată în cod de un grup `<g id="bd' . $db . '" op="1" ...>` translat pe axele x și pe y conform algoritmului de dispunere a informațiilor privitoare la bazele de date. În interiorul grupului, vom avea codul de desenare propriu-zisă a bazei de date (tabele și relații între ele). Atributul `op` semnifică operația pe care putem să o efectuăm asupra sa. Implicit are valoarea 1, adică o putem maximiza, iar în momentul în care este minimizată vom seta `op` pe 0, adică vom avea de efectuat operația de minimizare.

La maximizare, baza de date trebuie translată de la poziția inițială la poziția (0, 0), astfel încât să ocupe întreaga suprafață. Totodată, se face și o scalare, pentru a-i mări dimensiunile (la maximizare) sau a i le micșora (la minimizare).

```
fwrite($file, '<animate attributeName="viewBox"
  from="0 0 ' . $height . ' ' . $height . '" to="0 0 ' .
  (4*$height) . ' ' . (4*$height) .
  '" fill="freeze" begin="indefinite" dur="1s"/>
  <animate attributeName="viewBox" from="0 0 ' .
  (4*$height) . ' ' . (4*$height) . '" to="0 0 ' .
  $height . ' ' . $height .
  '" fill="freeze" begin="indefinite" dur="1s"/>');
```

Scalarea se realizează cu ajutorul atributului `viewBox`, dimensiunile mărindu-se sau micșorându-se de 4 ori, dar la fel de bine se poate realiza și printr-un `transform="translate(0.25)"`.



Vizualizarea în fereastra client a bazelor de date

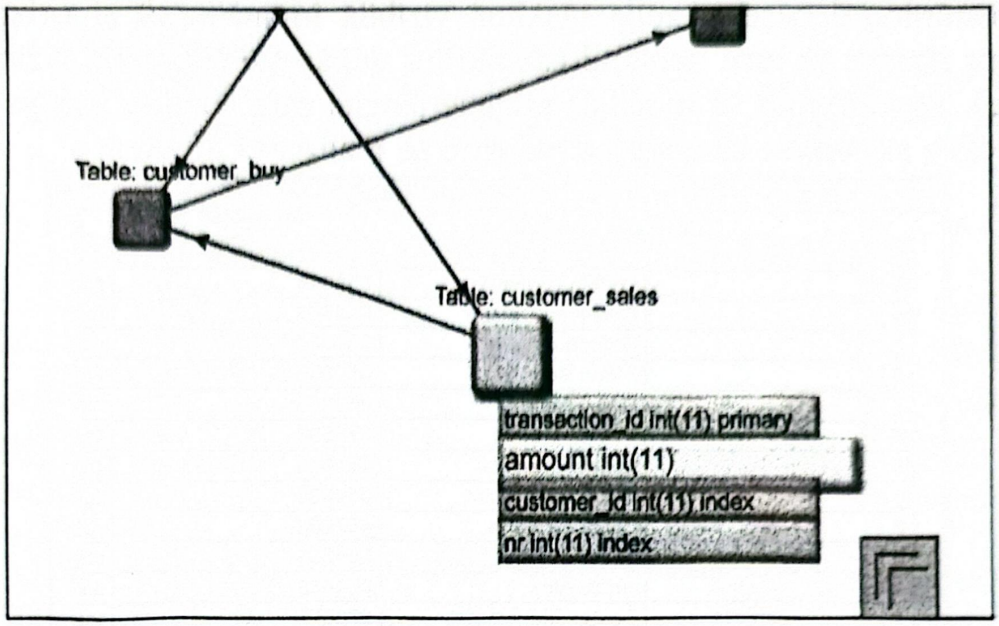
O schemă a apelurilor imbricate ce determină desenarea anterioară este:

```
for($db = 0; $db < $nr_db; $db++)
{
  fwrite($file, '<rect id="background' . $db .
    '" x="2" y="2"  width="' . ($height-4) .
    '" height="' . ($height-4) .
    '" fill="white" opacity="0.2">');
  //desenam legaturile din baza de date db
  drawPaths($db);
  //desenam tabelele din baza de date db
  drawTables($db); //aici se apeleaza drawFields(),
                   //in care se invoca drawRecords())
}
```

O dată maximizată o bază de date, utilizatorul poate interacţiona cu aceasta, putând vedea tabelele care o compun, relaţiile dintre acestea, câmpurile conţinute în tabele şi înregistrările din tabele. Funcţia `drawTables($db)` este cea responsabilă cu dispunerea circulară a tabelelor, `drawPaths($db)` – cu desenarea legăturilor dintre tabele, `drawFields($db, $table)` – cu desenarea câmpurilor şi `drawRecords($db, $table, $field)` – cu desenarea înregistrărilor.

Rezultatul desenării câmpurilor pentru fiecare tabelă este următorul: funcţia `drawFields()` determină dispunerea grafică a numelor câmpurilor sub tabela de care aparţin, sub forma unui mic meniu, din care putem selecta

unul dintre ele pentru a obține vizualizarea înregistrărilor sortate crescător după acel câmp:



Vizualizarea câmpurilor pentru tabela *customer_sales*

Se observă animația realizată: la situarea mouse-ului peste numele unei tabele, va apărea lista de câmpuri după ce, în prealabil, mărim dimensiunea tabelei și în aplicăm un efect de umbră; la plasarea mouse-ului deasupra unui câmp, acesta se selectează prin mărirea dimensiunilor, schimbarea culorii și aplicarea aceluiași efect de umbră.

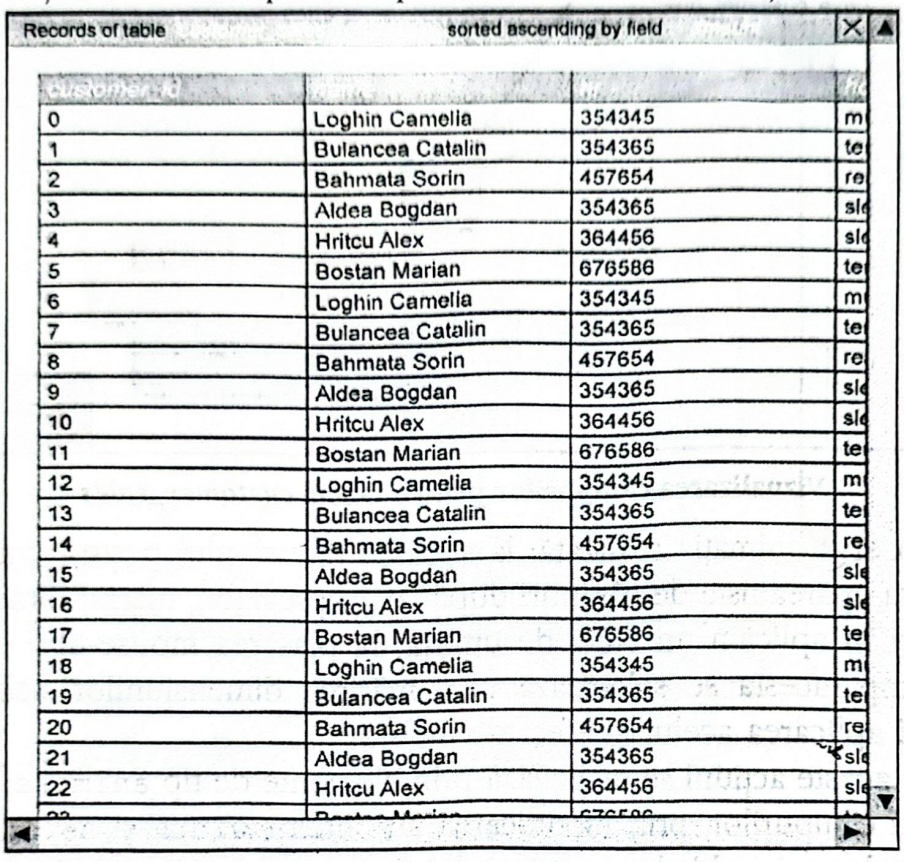
Toate aceste acțiuni se realizează prin elemente de tip `animate`: în cazul tabelei și câmpurilor, prin modificarea atributelor `width` și `height`, iar în cazul textelor, prin schimbarea atributului `font-size`, realizată prin elementul:

```
<animateTransform
  attributeName="transform" type="scale" values="1;1.2"
  begin="field1' . $db . '_' . $id . '_' . $i .
  '. mouseover" dur="0.5s" repeatCount="1"
  additive="replace" calcMode="linear" fill="freeze"/>
```

în care tipul transformării este `scale`, adică scalare, care se va realiza de la dimensiunile normale la o creștere cu 0.2 : `values="1;1.2"`.

La apăsarea unui câmp, vor apărea înregistrările acelei baze de date, sortate în ordine crescătoare după denumirea câmpului apăsat. Această parte este una de construire dinamică a înregistrărilor. Dacă până acum aveam deja desenate toate obiectele care compun aplicația (baze de date, tabele, legături) și pentru a le vizualiza pe rând făceam o ascundere a celorlalte (prin modificarea atributului `visibility` la `hidden`), de această dată creăm efectiv la apăsarea mouse-ului înregistrările prin metode JavaScript, folosind modelul DOM: `createElement()`, `setAttribute()`, `appendChild()`.

Vom simula o fereastră normală cu bară de titlu, buton de închidere a ferestrei şi bare de defilare verticală (pentru cazul în care numărul înregistrărilor depăşeşte zona maximă vizibilă în înălţime) şi orizontală (pentru situaţia în care numărul câmpurilor depăşeşte suprafaţă vizibilă în lăţime). Bara de titlu va specifica numele tabelei care conţine înregistrările respective şi numele câmpului după care se realizează afişarea:

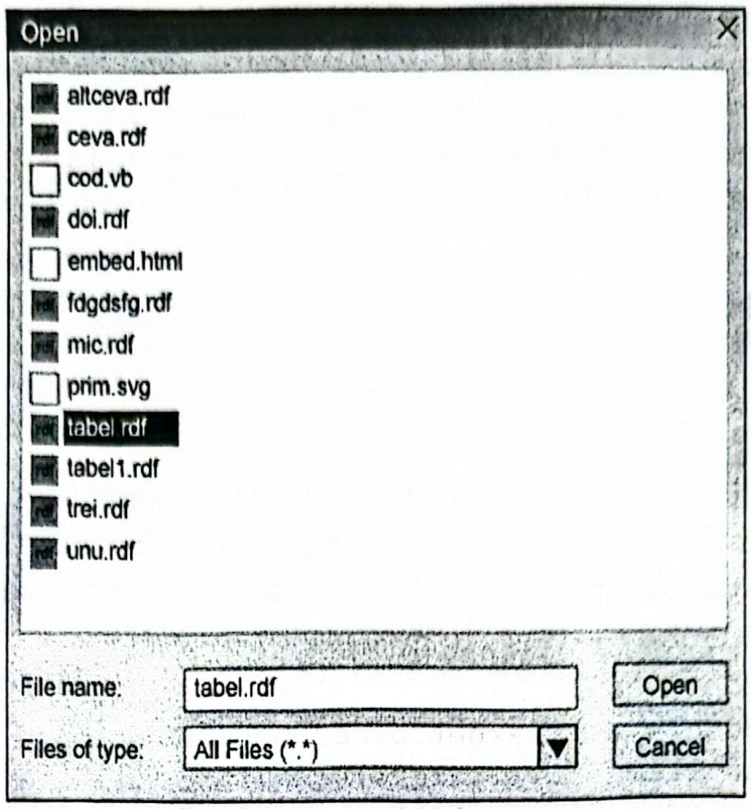


customer_id			
0	Loghin Camelia	354345	m
1	Bulancea Catalin	354365	te
2	Bahmata Sorin	457654	re
3	Aldea Bogdan	354365	sl
4	Hritcu Alex	364456	sl
5	Bostan Marian	676586	te
6	Loghin Camelia	354345	m
7	Bulancea Catalin	354365	te
8	Bahmata Sorin	457654	re
9	Aldea Bogdan	354365	sl
10	Hritcu Alex	364456	sl
11	Bostan Marian	676586	te
12	Loghin Camelia	354345	m
13	Bulancea Catalin	354365	te
14	Bahmata Sorin	457654	re
15	Aldea Bogdan	354365	sl
16	Hritcu Alex	364456	sl
17	Bostan Marian	676586	te
18	Loghin Camelia	354345	m
19	Bulancea Catalin	354365	te
20	Bahmata Sorin	457654	re
21	Aldea Bogdan	354365	sl
22	Hritcu Alex	364456	sl

Vizualizarea înregistrărilor unei tabele

Se observă că numărul de înregistrări depăşeşte atât în lăţime, cât şi în înălţime fereastra de vizualizare, deci vom avea activate cele două bare de *scroll*, verticală şi orizontală. La apăsarea butonului de închidere X, acestea vor fi doar ascunse, deci vom modifica valoarea atributului `visibility` din `visible` în `hidden`, pentru ca a doua oară când vom apăsa pe acelaşi câmp să nu mai pierdem timp cu crearea lor, ci doar să le facem din nou vizibile. Acest tip de implementare măreşte viteza de execuţie a aplicaţiei.

O altă realizare grafică este cea a ferestrelor de dialog *Open* şi *Save*. De fapt, avem doar o singură implementare pentru ambele obiecte; diferă doar titlul şi eticheta butoanelor de încărcare, respectiv salvare, precum şi operaţiunile efectuate cu aceste butoane. Vom dispune de o fereastră tipică de încărcare/salvare, cu posibilitatea selectării fişierului care dorim să fie deschis, a scrierii într-un *text-box* a numelui fişierului pe care dorim să-l salvăm şi a suprascrierii (*overwrite*). Rezultatul este următorul:



Fereastra de dialog *Open* pentru încărcarea de fişiere .rdf

Se observă că putem face o vizualizare a tuturor fişierelor din directorul `./saved`, care pot fi sau nu fişiere cu extensia *.rdf*. Dacă sunt documente *RelDF*, atunci le vom ataşa iconul corespunzător, altfel nu recunoaştem formatul.

Fereastra de dialog nu este o componentă predefinită, neexistând aşa ceva în SVG. Întreaga interfaţă este construită în cod, linie cu linie, prin elemente de tip `<rect>` şi `<text>`.

Dacă încercăm deschiderea unui document care nu are extensia *.rdf*, vom afişa un mesaj de eroare de genul *„Cannot open file!"*.

Fereastra de mai sus este construită în cod SVG astfel: vom defini un grup `<g id="file_open_save_form">`, care va conţine elementele ferestrei de dialog. Asupra sa definim mai multe evenimente: `onkeypress`, `onmousedown`, `onmouseup` pentru a putea selecta un document, pentru a putea scrie în *text-box*, pentru a putea face *drag-and-drop* asupra ferestrei etc. La apăsarea butonului *Open* sau *Cancel*, se va ascunde fereastra cu ajutorul elementului `<set>`:

```
<set attributeName="visibility" to="hidden"
 begin="open_save_form_cancel.click;Xfile_open_save.click"
/>
```

Modulul JavaScript

Această parte este cea răspunzătoare de interacţiunea utilizatorului cu aplicaţia via interfaţa grafică. Fiecare control din aplicaţia noastră trebuie să

aibă o utilitate, să răspundă într-un fel la diferitele acţiuni pe care le putem efectua asupra lor. De aceea, vor fi tratate evenimentele de `click`, `mouseover`, `mousepress`, `mouseup`, `mousemove`, `load` etc.

În primul rând, să vedem cum funcţionează meniul. La apăsarea fiecărui element din meniu (*Document*, *Databases*, *Help*), se va trata evenimentul *click* astfel:

```
fwrite($file, '<rect id="document" op="expand" x="0" y="0"
  width="' . ($clientX-40) . '" height="' .
  $item_height .'" opacity="0"
  onclick="ExpandCollapse(evt)" />');
```

Acesta este exemplul pentru opţiunea *Document*. Se observă că, la apăsarea elementului `<rect>` cu identificatorul `document`, se apelează metoda `ExpandCollapse(evt)`. Să aruncăm o privire asupra sa:

```
function ExpandCollapse(evt)
{
     svgdoc = evt.target.ownerDocument;
     menu_item = evt.target;
     id = menu_item.getAttribute("id");
     operation = menu_item.getAttribute("op");
     //determinam id-ul itemului de meniu
      //pe care am dat click
     switch(id)
     {
          case "document":  //optiunea este Document
           ...
           break;
           case "databases": //optiunea este Databases
            ...
           break;
           case "help":      //optiunea este Help
            ...
           break;
       }
 }
```

Se determină rădăcina documentului SVG cu care interacţionăm (`document_element`) prin `svgdoc=evt.target.ownerDocument`. De asemenea, se află identificatorul opţiunii de meniu, pentru a distinge între ele: `id = menu_item.getAttribute("id")`, precum şi operaţia pe care urmează să o realizăm (extindere ori restrângere): `operation= menu_item.getAttribute("op")`. Urmează `switch(id)`, care tratează cazurile corespunzătoare fiecărei opţiuni de meniu în parte. În fiecare caz, se pornesc animaţiile pentru coborârea sau ridicarea elementelor de sub-meniu, pentru transformarea vizibilităţii.

De exemplu, pentru a anima coborârea meniului *Document*, vom porni animaţia ce determină mişcarea dreptunghiului alb, precum şi a textelor şi a desenelor de pe acesta.

```
if(childs.item(i).nodeName == "rect" &&
    childs.item(i).getAttribute("height") == 0) {
  if(operation == "expand")
    animates[0].beginElement();
  else          // adica operation == collapse
    animates[1].beginElement();
}
```

Se va testa deci dacă avem de-a face cu o operaţie de extindere sau de restrângere. În primul caz, se porneşte animaţia care iniţial avea atributul `begin="indefinite"`.

O altă funcţie importantă este `OnLoad(evt, nr_db, dim_db, offset, client_w, client_h, client_x)` care se apelează la încărcarea documentului SVG, deci tratează evenimentul `onload`. Ea verifică dacă la încărcarea documentului se depăşeşte sau nu înălţimea ferestrei de afişare a bazelor de date şi, în caz afirmativ, activează *scroll*-ul vertical:

```
db_trans_y = rows*dim_db + (rows+1)*offset;
//daca translarea pe y e mai mare decat
//inaltimea zonei client, introducem scroll-ul
if(db_trans_y > client_h)
{
     path_up = svgdoc.getElementById("path_scroll_up");
     path_down =
          svgdoc.getElementById("path_scroll_down");
     path_up.setAttribute("opacity", "1");
     path_up.setAttribute("many_db", "1");
     path_up.setAttribute("scroll_what", "db");
     path_down.setAttribute("opacity", "1");
     path_down.setAttribute("many_db", "1");
     path_down.setAttribute("scroll_what", "db");
}
```

Activarea barei de defilare se face prin modificarea atributului `opacity` pentru cele două butoane care conţin *scroll*-ul, `path_up` şi `path_down`. Vom verifica şi ce anume parcurgem: baze de date sau înregistrări.

La poziţionarea mouse-ului deasupra unui tabel, se observă apariţia câmpurilor din acel tabel. În `TableMouseOut(evt, dbase, id, offset)` şi `FieldMouseOut(evt, dbase, id, field, offset)` se verifică dacă am rămas sau nu în interiorul acelor câmpuri. Acest lucru se realizează prin funcţiile de determinare a coordonatelor din zona client: `evt.getClientX()` şi `evt.getClientY()`.

Aflăm, mai întâi, coordonatele primului câmp, o dată ce am ieşit din tabel. Dacă mouse-ul e plasat deasupra acestui câmp, înseamnă că nu facem invizibile câmpurile, pentru că suntem încă în interiorul tabelului. Iată cum arată determinarea zonei în care se află pointer-ul mouse-ului:

```
if(currentX >= x1_f && currentX <= x2_f &&
   (currentY+2) >= y1_f && currentY <= y2_f)
{
     first_field_out = false;
}
```

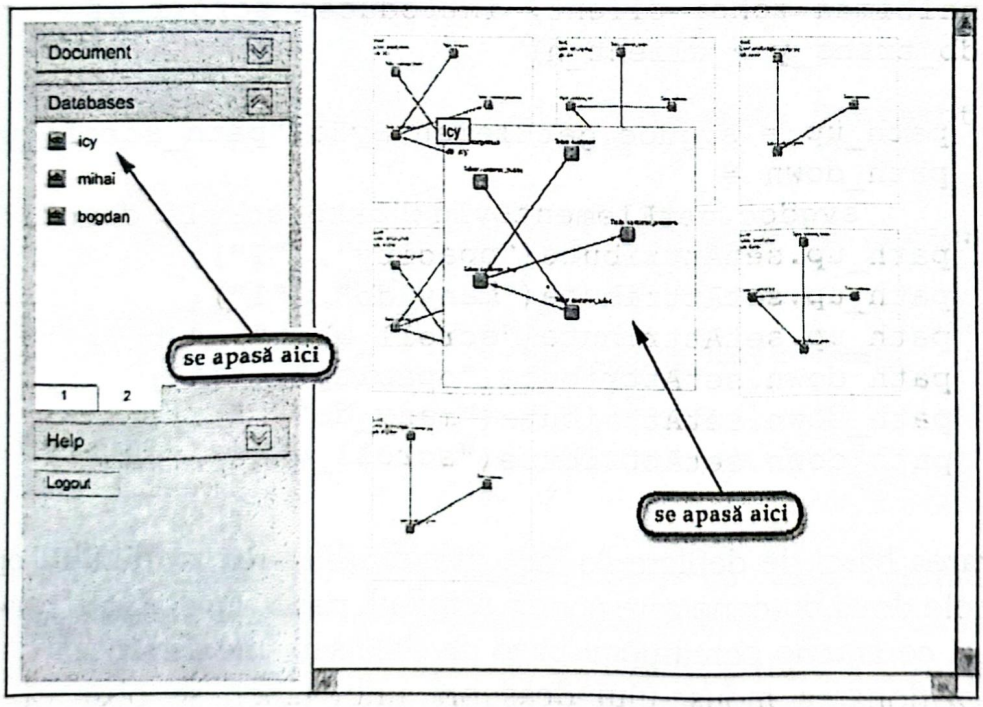
unde `x1_f`, `x2_f`, `y1_f` şi `y2_f` reprezintă coordonatele primului câmp.

O operaţie importantă este cea de maximizare/minimizare a bazelor de date. Aceasta se poate realiza în două moduri:

- prin meniu (în care, dacă apăsăm mai întâi un subelement, vom maximiza baza de date, iar apoi, dacă apăsăm imediat acelaşi subelement, o vom minimiza);
- prin *click* în zona client pe baza de date pentru maximizare şi apăsarea butonului *minimize* pentru minimizare.

Metodele care implementează aceste operaţii sunt `Maximize(evt, dbase)`, `Minimize(evt, dbase)` şi `MaxMinClick(evt, dbase)`.

Iată cum funcţionează maximizarea unei baze de date:

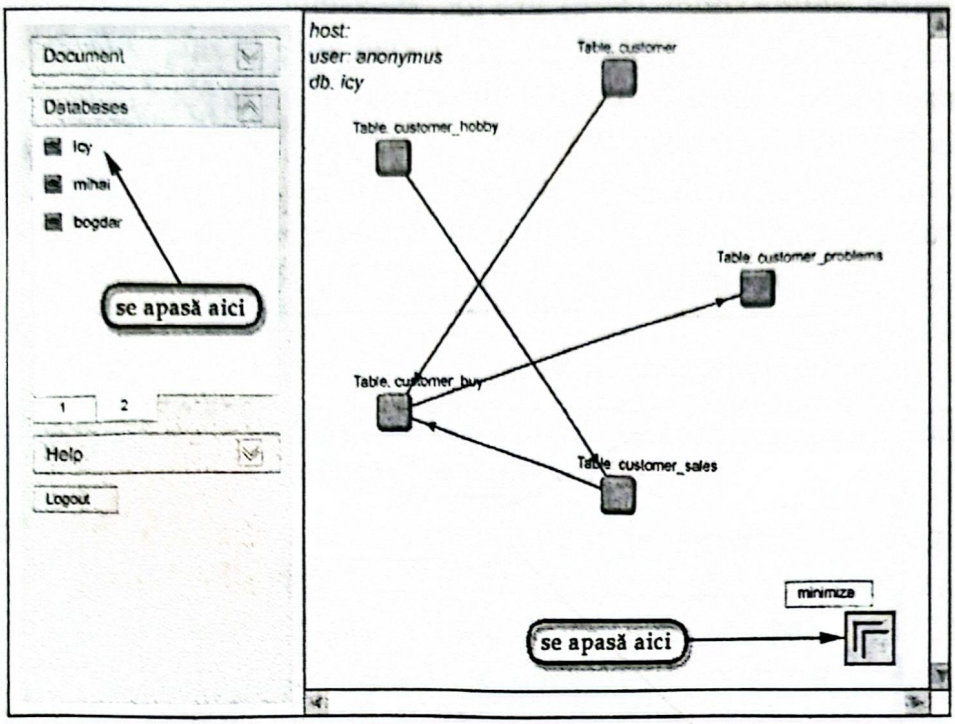


Maximizarea unei baze de date

Operaţia de maximizare este puţin mai complicată, pentru că este posibil ca, la apăsarea unui subitem din meniu, să existe deja o bază de date maximizată. Atunci, trebuie, mai întâi, să o minimizăm pe cea anterioară, şi abia apoi să o maximizăm pe cea dorită. Acest lucru se realizează astfel: se

parcurg toate grupurile <g> care au ID-ul de tipul bdX, unde X este numărul de identificare al fiecărei baze de date. Acesta va conține un atribut op, care ne dă informația necesară pentru a afla ce operație putem efectua cu baza de date respectivă. Dacă op este 0, atunci baza de date este maximizată și trebuie minimizată, deci pornim animațiile corespunzătoare.

Modul de funcționare a minimizării unei baze de date este ilustrat în figura următoare:



Minimizarea unei baze de date

O funcționalitate interesantă este cea de realizare a operației de *drag-and-drop*. Astfel, vom putea mișca obiectele în interiorul ferestrei prin apăsarea și mișcarea mouse-ului. Acest lucru se realizează prin tratarea evenimentelor mousedown, mousemove și mouseup asupra elementelor ce constituie tabelele, precum și asupra ferestrelor de dialog *Open* și *Save*.

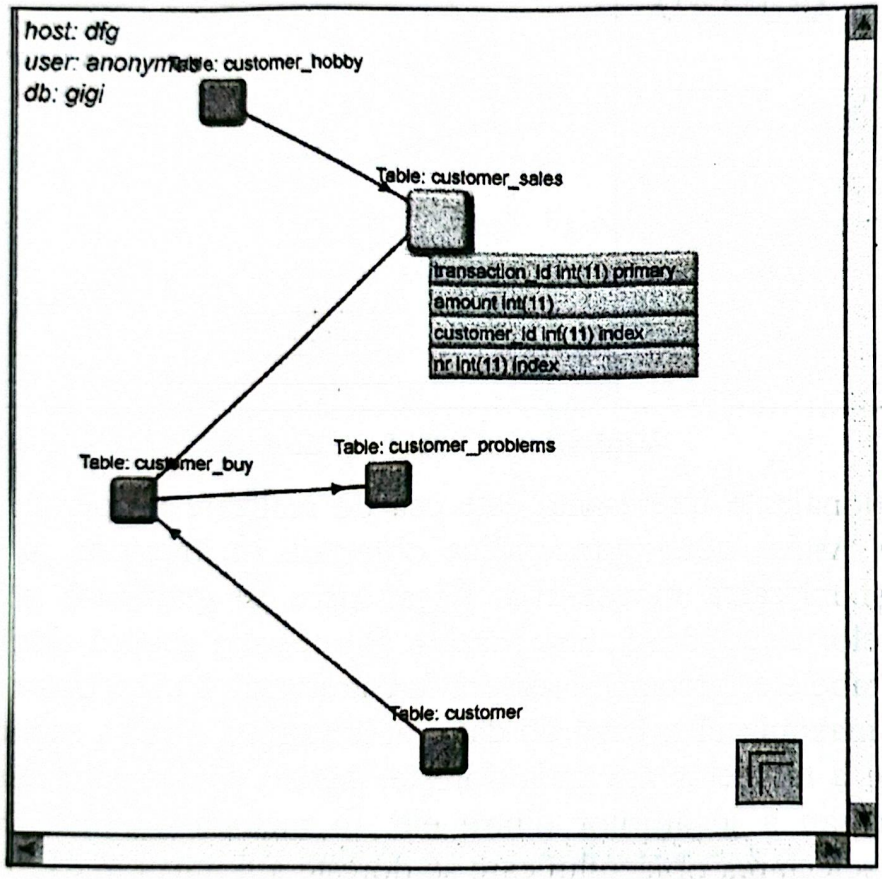
O dată maximizată o bază de date, utilizatorul va putea schimba modul de aranjare a tabelelor așa cum dorește, pentru o mai bună vizualizare a câmpurilor sau a legăturilor dintre ele. În metoda MouseDown(evt) se realizează selectarea obiectului care se dorește a fi mutat în cadrul ferestrei. Se efectuează apoi o verificare pentru a determina obiectul de mutat :

```
selected = evt.target;
id = selected.getAttribute("id");
if(id == "open_save_title_bar")
//am selectat fereastra de Open/Save
{
     moving = vgdoc.getElementById("file_open_save_form");
     selected = moving;
}
```

```
if(id.indexOf("tab" + this_dbase) == 0)
//adica am selectat o tabela din baza de date curenta,
//this_dbase
{
      moving = selected.parentNode;
      selected = moving;
}
```

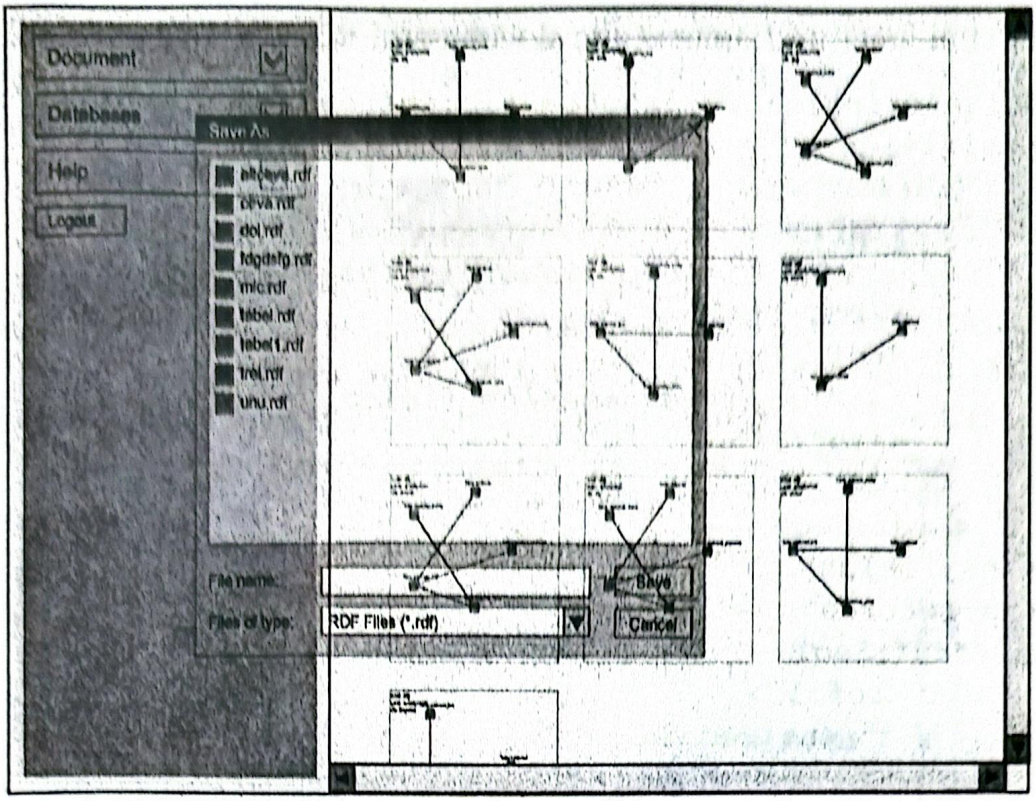
În metoda `MoveWindow(evt)` are loc, de asemenea, o verificare pentru a determina obiectul care se mișcă și apoi se stabilește poziția obiectului la noile coordonate ale pointer-ului mouse-ului, `nx` și `ny`: `selected.setAttribute ('transform', 'translate('+nx+','+ny+')')`.

Se realizează și o testare pentru a nu permite obiectelor mișcate să iasă din cadrul ferestrei. Iată noua aranjare a tabelelor unei baze de date după ce am utilizat *drag-and-drop* (vă reamintim că, inițial, dispunerea era în cerc):



O posibilă nouă aranjare a tabelelor unei baze de date

În cazul mișcării formei de *Open/Save*, vom realiza o reducere a transparenței:



Mutarea ferestrei de dialog

La apăsarea pe un câmp al unei tabele, vom genera dinamic înregistrările. Astfel, fiecare element `<rect>` corespunzător unui câmp va avea un atribut `onclick` care va lua valoarea `FieldClick(evt, '.$db.', '.$id.', '.$i.', '.$x_col.' ,'.$y_col.')`. Metoda amintită va construi înregistrările pentru câmpul identificat de baza de date `$db`, tabela `$id`, identificatorul de câmp `$i` şi coordonatele câmpului, `$x_col` şi `$y_col`.

Pentru a determina înregistrările, va trebui să executăm un script PHP care va genera un document RDF prelucrat apoi de aplicaţie. Documentul creat va fi încărcat într-un arbore DOM pentru a putea fi parcurs, şi acest lucru se realizează prin crearea unui obiect de tip ActiveX (oferit de navigatorul Internet Explorer). Iată codul sursă:

```
var xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async = "false";
if(!xmlDoc.load("http://localhost/licenta/sib/getRDF.
        php?h=" + host + "&u=" + user + "&p=" + password +
        "&db=" + dbase + "&table=" + table + "&field=" +
        field)) {
    alert("Could not load records from table");
    exit(1);
}
```

Scriptul *getRDF.php* primeşte argumentele necesare pentru a executa cererea, şi anume crearea documentului RDF ce va fi prelucrat.

Iată cum arată un fragment din documentul RDF creat de scriptul PHP:

```
<rdf:Description rdf:about="sib:customer:customer_id">
  <ns:fields>
     <rdf:Seq>
        <rdf:li>
           <ns:field>customer_is</ns_field>
        </rdf:li>
        <rdf:li>
           <ns:field>name</ns_field>
        </rdf:li>
        ...
     </rdf:Seq>
  </ns:fields>
  <ns:records>
     <rdf:Seq>
        <rdf:li>
           <ns:record>
              <rdf:Seq>
                 <rdf:li>0</rdf:li>
                 <rdf:li>Loghin Camelia</rdf:li>
                 ...
              </rdf:Seq>
           </ns:record>
        </rdf:li>
        ...
     </rdf:seq>
  </ns:records>
</rdf:Description>
```

Următorul pas este determinarea numelor, precum și a numărului de câmpuri și de înregistrări din tabelă:

```
fields = xmlDoc.getElementsByTagName("ns:field");
nr_fld = fields.length;
records = xmlDoc.getElementsByTagName("ns:record");
nr_rec = records.length;
```

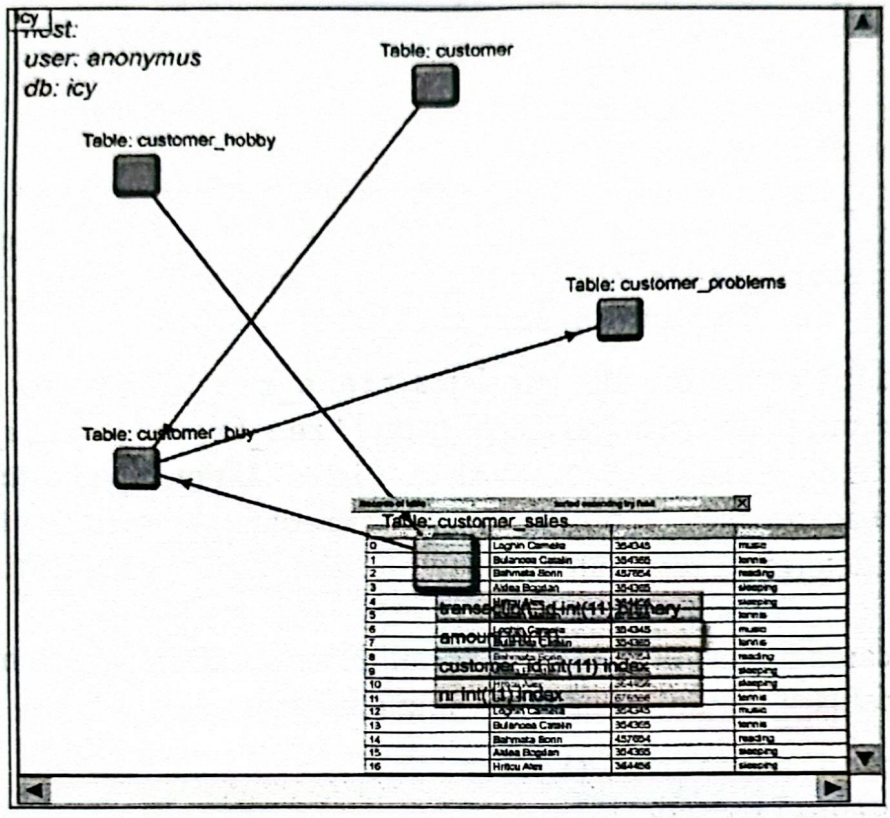
Este momentul în care folosim metodele JavaScript pentru DOM de parcurgere a arborelui creat și de creare de elemente SVG pentru desenarea dinamică a înregistrărilor. Urmează un exemplu de creare a unui element <rect>:

```
node = root.createElement("rect");
    node.setAttribute("x", j*r_width);
node.setAttribute("y", 20);
node.setAttribute("width", r_width);
node.setAttribute("height", r_height);
```

```
node.setAttribute("style",
   "stroke:black;stroke-width:1;fill:gray;opacity:0.4");
record_trans.appendChild(node);
```

Elementul va fi creat cu ajutorul metodei `createElement("rect")`, după care va primi atributele dorite cu `setAttribute()`, iar, la sfârșit, va fi adăugat cu `appendChild(node)`, ca fiul elementului părinte dorit.

Înregistrările vor fi create la coordonatele câmpului cu o dimensiune de 4 ori mai mică decât cea normală – `scale(0.25)` – și vor fi mărite și translate în același timp, simulând astfel maximizarea unei ferestre.



Crearea înregistrărilor la apăsarea unui câmp (remarcați realizarea transparenței)

Înregistrările își pornesc mișcarea de la coordonatele câmpului *amount* și vor ajunge să ocupe întreaga fereastră client, în timp ce baza de date va fi ascunsă.

Tot aici, în metoda `FieldClick()` se va apela o altă funcție importantă, `OnLoadRecords(evt, nr_records, nr_columns, padding, record_h, record_w, client_h)`, care primește ca parametri numărul de câmpuri și de înregistrări determinate anterior, înălțimea și lățimea unei înregistrări, precum și ale zonei client. În cadrul acestei metode, se verifică dacă se depășește atât în lățime, cât și în înălțime fereastra client și, în caz afirmativ, se activează barele de defilare verticală și/sau orizontală.

Există două funcții de *scroll*, `v_scroll(v_dir)` pentru defilarea pe verticală și `h_scroll(h_dir)` pentru cea orizontală. În aceste două funcții se testează mai întâi dacă barele sunt activate (adică `path_left/right/`

up/down.getAttribute ("opacity") == 1), iar în caz afirmativ, se calculează gradul de translare pe axa *x* sau pe *y*. Apoi, se atribuie această translare grupului conținând înregistrările ori bazele de date:

```
h_Dir=h_dir;
dif = rec_trans_x - client_height;
if(h_dir == 1 && h_disp_rec > -dif) { // stanga
    h_disp_rec -= 25;
}
else
  if(h_dir == 2 && h_disp_rec < 0) { // dreapta
    h_disp_rec+=25;
}
trans_rec_h = "translate(" + h_disp_rec + ","
  + v_disp_rec + ")";
record_trans = svgdoc.getElementById("rec_trans" +
  this_dbase + "_" + this_table + "_" + this_field);
    record_trans.setAttribute("transform",trans_rec_h);
loop = setTimeout('h_scroll(h_Dir)', 50);
```

Deci, după ce am calculat translarea `trans_rec_h`, vom asocia această translare elementului <g> având identificatorul `"rec_trans" + this_dbase + "_" + this_table + "_" + this_field`. Ultimul pas este apelarea recursivă a acestei metode prin `loop = setTimeout('h_scroll (h_Dir)', 50)`, care determină reapelarea acestei metode la o distanță în timp de 50 de ms.

La ridicarea mouse-ului de pe butonul de *scroll*, se va invoca metoda de oprire a defilării. Ea are următoarea implementare:

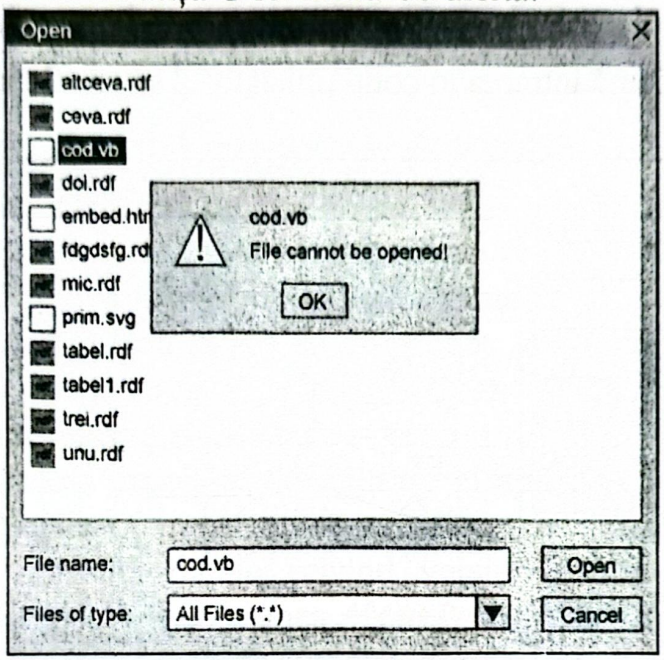
```
function stopscroll() {
    clearTimeout (loop);
}
```

Metoda `MenuOpenSave(evt)` va determina deschiderea ferestrei de dialog *Open* (sau *Save*). Fereastra de dialog se desenează în PHP dacă parametrul `open` (sau `save`) trimis via URL este setat pe 1. La apăsarea în meniu a unei opțiuni, acest parametru va lua valoarea 1:

```
RDFfile = evt.target.getAttribute("file");
location = "http://localhost/licenta/RelVis.php?h=" +
  host + "&u=" + user + "&p=" + password +
  "&login=0&fromSaved=" + fromSaved + "&" +
  dialogTypes[parseInt(dlg_type)] + "=1&file=" +
  RDFfile;
```

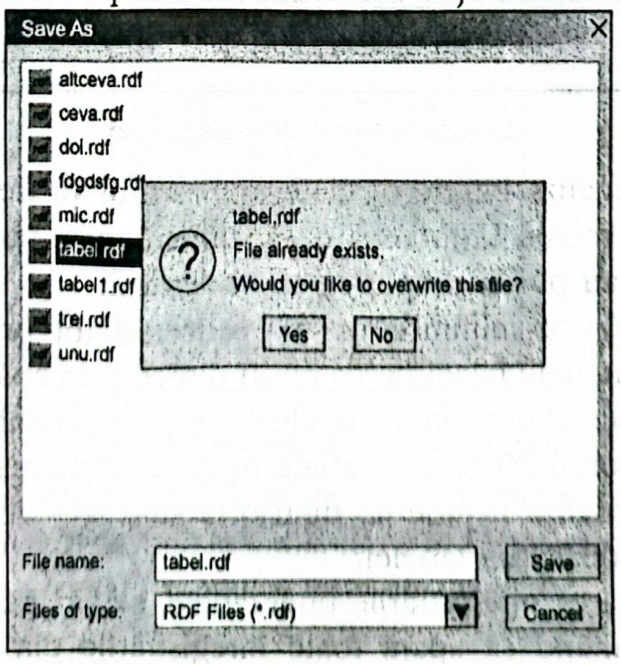
Aceste linii de cod vor determina executarea scriptului *RelVis.php* cu argumentele corespunzătoare. Variabila `location` este o proprietate de bază a obiectului predefinit `window`, care oferă informații despre URI. Se observă că `dialogTypes[parseInt(dlg_type)]` ne indică dacă este vorba de fereastra *Open* sau *Save*.

Metoda `FormOpenSave(evt, dlg_type)` este cea care lansează același script *RelVis.php*, cu parametrii depinzând de operația pe care dorim să o efectuăm: încărcarea sau salvarea unui document. Dacă acesta nu este în format *.rdf*, atunci se va afișa o fereastră de alertă:



Încercarea de deschidere a unui document nepermis

Este posibilă și afișarea unei ferestre de confirmare (*Confirm*), în cazul în care se dorește salvarea peste un document deja existent:

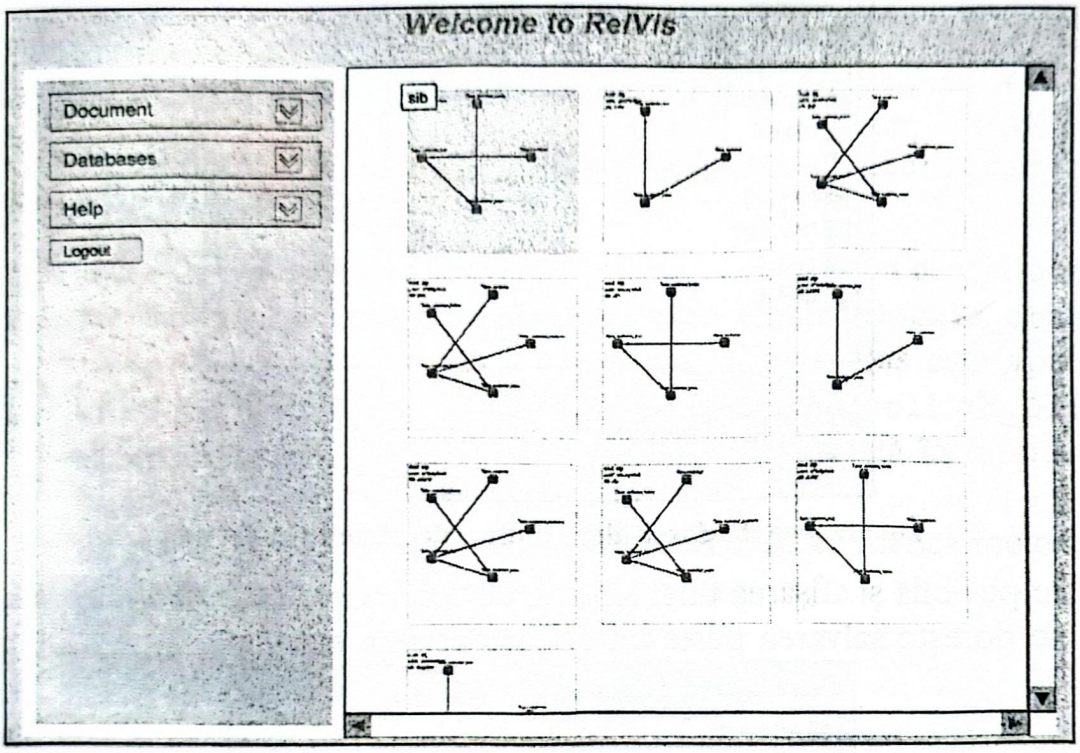


Afișarea ferestrei de confirmare în cazul încercării de salvare a unui document

Aceste două ferestre sunt generate tot dinamic prin apeluri de funcţii `createElement()`, `setAttribute()` şi `appendChild()` în metodele `Alert()` şi `Confirm()`. Invităm cititorii să le implementeze.

3.3 Manual de utilizare

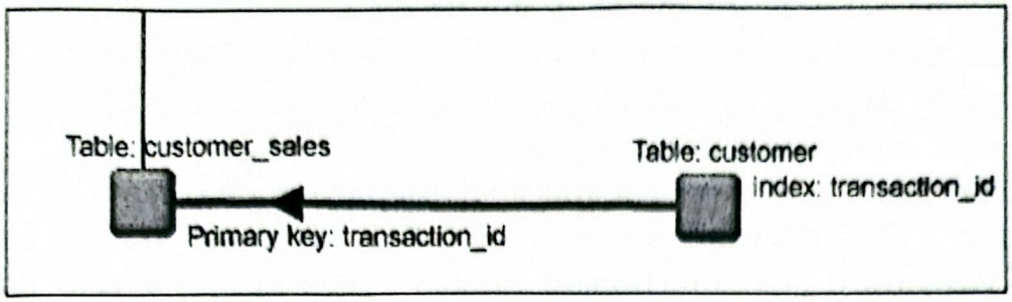
- Pentru intrarea în aplicaţie, utilizatorul se va afla în fereastra de autentificare.
- Vor trebui introduse următoarele date: *gazda* (*host*), *numele de utilizator* (*user*) şi *parola* (*password*). Acestea corespund cu datele de intrare necesare accesării unui server de baze de date MySQL (vezi capitolul anterior). După intrarea în cont, utilizatorul se va afla în faţa următoarei interfeţe:



Interfaţa generală a aplicaţiei

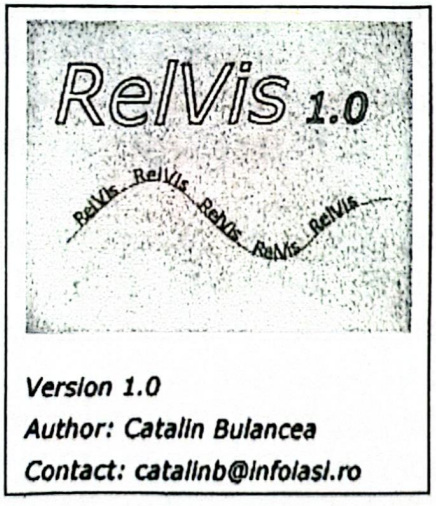
- Pentru a vizualiza bazele de date, utilizatorul va trebui să folosească opţiunea de meniu *Databases* (dacă sunt multe, se vor acţiona *tab*-urile care apar) sau pur şi simplu *scroll*-ul vertical.
- Maximizarea sau minimizarea se realizează tot prin apăsarea unei opţiuni cu numele bazei de date sau direct, prin apăsarea pe baza de date corespunzătoare în zona client (în cazul minimizării, se poate folosi şi butonul *Minimize*), aflată în colţul din dreapta jos.
- Pentru a vizualiza câmpurile dintr-o tabelă, se va poziţiona cursorul mouse-ului deasupra tabelei.
- Pentru a încărca înregistrările unei tabele, se va apăsa pe unul dintre câmpuri, urmând să apară toate înregistrările din tabelul respectiv sortate în ordine crescătoare după acel câmp.

- Pentru a vizualiza relaţiile dintre tabele, se va poziţiona cursorul mouse-ului deasupra unei linii, urmând să se afişeze indecşii şi cheile primare pe care le referă:



Vizualizarea relaţiilor dintre tabele

- Pentru a muta tabelele sau ferestrele de dialog prin *drag-and-drop*, se va apăsa butonul stâng al mouse-ului şi se va mişca ţinând apăsat până când se ajunge la poziţia dorită de utilizator.
- Pentru a încărca (sau salva) un document *.rdf*, se va accesa opţiunea *Document*, după care şi opţiunea *Open* (sau *Save*), determinând deschiderea ferestrelor de dialog corespunzătoare.
- Pentru a accesa manualul de utilizare al aplicaţiei, se va selecta din meniu opţiunea *Contents* sau se va apăsa tasta F1.
- Pentru a consulta datele privind versiunea aplicaţiei şi numele autorului, se va selecta opţiunea *About RelVis*, care va activa următoarea fereastră SVG (se va închide automat la apăsarea pe suprafaţa acesteia):



Vizualizarea ferestrei *About RelVis*

- Pentru a ieşi din aplicaţie, se va apăsa butonul *Logout* din meniu; acţionarea acestuia duce la întoarcerea în fereastra de autentificare, cu posibilitatea accesării contului unui alt utilizator.

3.4 Testare și configurare

Am testat aplicația pe o mașină cu un procesor având frecvența de 1.2 GHz și memorie RAM de 384 MB. Se pare că resursele procesorului sunt ocupate în întregime, iar aplicația merge destul de greu.

O altă testare s-a realizat pe o mașină Pentium IV cu frecvența de 2.8 GHz și memorie RAM de 512 MB. Aplicația este destul de rapidă, însă și aici procesorul este ocupat în mare parte.

În concluzie, pentru rularea aplicației este nevoie de o stație de lucru de cel puțin 1 GHz și având memoria internă de cel puțin 512 MB pentru o bună vizualizare și o funcționare rapidă. *Plug-in*-ul SVG este un mare consumator de resurse.

Pentru a rula aplicația, este necesară instalarea pe partea client a unui *plug-in*, disponibil gratuit și numit *SVGviewer*, care poate fi găsit la adresa *http://www.adobe.com/svg/viewer/install/*. Se realizează apoi copierea direct pe un server Web (Apache sau IIS) a directoarelor și fișierelor aplicației (scripturi PHP și JavaScript) și acordarea drepturilor de acces depline directorului `./saved`, pentru a se putea realiza salvări de documente în el.

4. Comentarii bibliografice

Înainte de toate, este necesară studierea limbajului care stă la baza SVG, de aceea *XML Bible (Second Edition)*, Hungry Minds, 2001 de Elliote Rusty Harold este un bun punct de plecare. O altă referință poate fi *XML Black Book*, Coriolis Group, 1998 de Natanya Pitts-Moultis, care face o descriere destul de amănunțită a familiei XML. De asemenea, se poate vizita situl *http://www.w3.org/XML/* pentru a se parcurge specificația XML redactată de Consorțiul Web.

Pentru partea aplicației care se referă la programarea *server-side*, cititorul poate găsi în manualul PHP oficial o foarte bună referință vizitând adresa *http://www.php.net/*. A se parcurge, de asemenea, și precedentele capitole ale materialului de față.

Pentru partea aplicației care se referă la programarea *client-side*, se poate consulta cartea *JavaScript Bible*, IDG Books, 1998, de D. Goodman, sau un ghid complet situat pe Web la adresa *http://devedge.netscape.com/library/manuals/2000/javascript/1.3/guide*. Alte adrese de interes sunt *http://www.javafile.com/* și *http://www.js-examples.com/*. Informații utile privitoare la ECMAScript, varianta standardizată a limbajului JavaScript, sunt disponibile la *http://www.ecma.ch/*.

În sfârșit, pentru informații referitoare la SVG, care face, de fapt, subiectul acestui capitol, a se vedea specificația SVG W3C din ianuarie 2003 la adresa

http://www.w3.org/TR/SVG11/ sau cea a lui J. Bowler, *SVG 1.0 Specification*, W3C, 2001. O altă carte importantă este *Learn SVG: The Web Graphics Standard*, de Jon Frost, Stefan Goessner și Michel Hirtzler, apărută în anul 2002. Detalii privitoare la filtrele grafice SVG sunt date în capitolul cinci al volumului S. Buraga (coord.), *Aplicații Web la cheie*, Polirom, Iași, 2003. Mecanismul de realizare a animațiilor fiind similar celui din SMIL, cititorii interesați pot consulta și cartea M. Brut, S. Buraga, *Prezentări multimedia pe Web*, Polirom, Iași, 2004.

Pentru exemple, demo-uri și aplicații SVG extrem de folositoare, se pot vizita siturile *http://pilat.free.fr* și *http://adobe.com/svg/demos/main.html.* De asemenea, la adresa *http://www.carto.net/* puteți găsi articole, proiecte realizate în SVG și noutăți cu privire la acest limbaj, printre care și o legătură către *SVG Open Conference*, care a avut loc la Tokio, în luna septembrie 2004: *http://www.svgopen.org/.*

CAPITOLUL VI

FindMyHome, o soluție *e-business* în domeniul imobiliar

Viorel Atofani, vio@webartwiz.com

Capitolul de față reprezintă o privire de ansamblu asupra unei idei de afaceri *on-line*, prin detalierea unei soluții practice de *e-business*, și anume un portal Web în domeniul imobiliar. După o trecere în revistă a caracteristicilor portalurilor de tip *e-commerce*, se detaliază modul de proiectare și implementare în PHP a aplicației *FindMyHome*, insistându-se asupra manierei de administrare *on-line* a acesteia. Din punctul de vedere al implementării, capitolul oferă o modalitate de procesare a fișierelor *CSV* (*Comma Separated Values*).

1. Introducere

Afacerile *on-line* (denumite și *e-business*) reprezintă și vor reprezenta punctul forte al Internetului în viitorul mai mult sau mai puțin apropiat. Actualmente, acest tip de afaceri este deja bine pus la punct în diferite state, depunându-se eforturi pentru aplicarea lui cu succes și la noi în țară, dificultatea majoră fiind adaptarea cu succes la nevoile clienților și ale producătorilor sau prestatorilor de servicii autohtoni.

Este știut faptul că, în desfășurarea unei afaceri, o foarte mare importanță o are stabilirea cu exactitate a domeniului de activitate și găsirea publicului țintă căruia i se adresează. Încercând să transpunem aceste principii în termenii unei afaceri *on-line*, putem afirma că un *portal Web* poate constitui un *e-business de succes*, întrucât portalurile sunt dedicate, în general, unui domeniu bine definit, reunind cât mai multe informații legate de acesta, și, de asemenea, unei categorii speciale de vizitatori.

Capitolul de față își propune să dezvolte o idee de implementare a unei astfel de soluții *e-business* în domeniul imobiliar prin prezentarea arhitecturii unui portal intitulat *FindMyHome*.

2. Premise și obiective

Spațiul WWW a cunoscut o evoluție impresionantă, acest aspect având drept consecință directă dificultatea localizării informației. Astfel, a apărut necesitatea dezvoltării unor instrumente puternice care să caute și să catalogheze informațiile disponibile *on-line*, accesul la acestea dorindu-se a fi mult mai ușor și mult mai rapid. Responsabilitatea îndeplinirii unei asemenea activități revine *motoarelor de căutare* care traversează periodic Web-ul pentru a ține o evidență cât mai clară asupra informațiilor disponibile în cadrul acestuia.

Din păcate, și totodată, din fericire, motoarele de căutare, încercând să fie cât mai cuprinzătoare, furnizează rezultate astfel încât să acopere cât mai multe domenii legate de criteriul de căutare. Aceasta conduce adesea la furnizarea a foarte multe informații (deseori irelevante), aspect care presupune o muncă în plus din partea utilizatorului pentru a tria datele oferite de motorul de căutare și a le alege pe cele concludente.

O soluție alternativă, destinată în special utilizatorilor care știu exact ceea ce vor să caute, o reprezintă așa-numitele *portaluri Web*, considerate ca fiind o particularizare a motoarelor de căutare, în sensul că portalurile sunt dedicate unui domeniu bine definit, reunind cât mai multe informații în legătură cu acesta. De asemenea, ele pun la dispoziția utilizatorului diverse metode de prezentare și localizare (căutare) a informațiilor pe care le dețin, incluzând uneori și un modul de *e-commerce*, ceea ce presupune implementarea comerțului electronic *on-line* pentru anumite produse și/sau servicii prezentate.

Prin intermediul portalului au de câștigat atât potențialii clienți, aceștia găsind și procurând mai rapid produsul sau serviciul căutat, cât și producătorii sau prestatorii de servicii, având posibilitatea de a-și prezenta oferta direct clienților vizați.

În domeniul imobiliar, există diverse companii care sunt disponibile și pe Web, mai mult prin intermediul unor pagini de prezentare. Astfel, un potențial client care ar fi interesat să caute un imobil într-o anumită locație va trebui să apeleze la un motor de căutare clasic (de exemplu, Google) pentru a găsi și beneficia de informațiile oferite (*on-line*) de aceste firme, deci va trebui să treacă exact prin fazele destul de dificile despre care menționam:

- în primul rând, va trebui să acorde foarte multă atenție cererii adresate motorului de căutare, pentru a nu fi pur și simplu „bombardat" cu informații de cele mai multe ori inutile;
- din rezultatele prezentate, va trebui să le aleagă pe cele care sunt într-adevăr de interes pentru el;

- urmează o verificare, o trecere în revistă a tuturor rezultatelor (*link*-uri) selectate, activitate consumatoare de timp, având în vedere că trebuie parcurs fiecare sit în parte și trebuie analizate informațiile găsite;
- în final, va urma să realizeze o comparare a ofertelor clasificate drept interesante și să se decidă, eventual, asupra uneia dintre ele.

Nu este de neglijat nici faptul că, în acest procedeu de căutare a informațiilor dorite, munca utilizatorului va fi îngreunată și de modalitățile diferite de structurare a siturilor care conțin informația dorită. Astfel, utilizatorii vor trebui să se adapteze fiecărei modalități de prezentare a informației, specifice siturilor vizitate. Ce-i drept, o parte a siturilor vizate s-ar putea să folosească unul dintre șabloanele de navigare clasice, caz în care localizarea datelor nu va mai ridica probleme. Alte situri, pentru a-și face o prezență *on-line* deosebită, utilizează metode mai puțin obișnuite pentru a-și prezenta informațiile, cerând indirect vizitatorilor să „exploreze" modalitățile de navigare în cadrul acestora. Un exemplu semnificativ în acest sens sunt acele situri care recurg din plin la animația pusă la dispoziție de Macromedia Flash, pentru a oferi o interfață cât mai interesantă și atrăgătoare. Trebuie însă stabilit un raport judicios ales între o prezentare atractivă – bazată pe animație – și facilitarea accesului la informație.

De asemenea, navigarea dificilă în cadrul unui sit poate fi cauzată și de proiectarea defectuoasă a acestuia, fapt care determină vizitatorul să devină confuz atunci când intenționează să găsească o anumită informație, neștiind ce zonă din structura sitului respectiv trebuie să acceseze.

Toate aceste inconveniente, și nu numai, ar putea contura obiectivul unei posibile idei de afacere *on-line* (de tip *e-business*), menite să ajute utilizatorul să își desfășoare activitatea mult mai ușor și rapid, fără prea multe bătăi de cap și timp pierdut. După cum bine se cunoaște, satisfacerea nevoilor clientului, creșterea ergonomiei și oferirea unor produse/servicii de o calitate mai bună constituie premisele de la care se pleacă în demararea unei afaceri.

Astfel, pentru a veni în ajutorul persoanelor care doresc să caute pe Internet informații privitoare la imobile, vom încerca să dezvoltăm un portal imobiliar care să faciliteze acest lucru, oferind exact informația de care asemenea persoane au nevoie, într-un mod cât mai rapid. În plus, în cadrul acestuia mai pot fi puse la dispoziție informații adiționale din domeniul imobiliar, de interes pentru vizitatori, cum ar fi:

- informații referitoare la legislația în domeniu: noi legi apărute, amendamente etc.;
- anunțuri de interes pentru vizitatorii care caută ceva anume;
- sfaturi din domeniul imobiliar;
- nu în ultimul rând, oferte speciale.

3. Proiectarea aplicației

Colectarea datelor

Prima etapă pe care o presupune realizarea unui portal Web este culegerea de informații referitoare la domeniul vizat, informații provenind din mai multe surse, și punerea lor la dispoziția persoanelor interesate, într-o manieră structurată.

Acțiunea de „recoltare" a informațiilor – primordială în dezvoltarea unui asemenea tip de *e-business* – va fi pusă în practică prin încheierea unor contracte cu diferiți agenți imobiliari. Astfel, după cum se va vedea și în partea dedicată administrării portalului, vom solicita agenților să furnizeze periodic informațiile de care avem nevoie pentru a le integra în aplicație, mai ales datele privind noutățile de pe piața imobiliară. Pentru a facilita procesul de integrare a acestor noutăți, agenții imobiliari vor trebui să ofere informațiile conform unui șablon prestabilit, adoptându-se un format reprezentat de un fișier de tip *CSV* (*Comma Separated Values*). Acest tip de fișier reține informațiile în format text obișnuit, distribuite pe linii. Fragmentele de informație sunt delimitate în cadrul unei linii prin intermediul unor caractere fixate anterior, în mod obișnuit fiind utilizate virgula sau caracterul TAB.

De ce s-a ales acest tip de format pentru manipularea datelor?

Inițial, s-a încercat găsirea unei soluții cât mai convenabile în furnizarea și manipularea informațiilor necesare pentru desfășurarea acestei activități, atât din punctul de vedere al agenților imobiliari, cât și din perspectiva administrării aplicației. Alegerea acestui tip de format dintre celelalte avute în vedere a fost motivată de faptul că agenții imobiliari folosesc destul de frecvent aplicații de calcul tabelar, cum ar fi Microsoft Excel sau Open Office, iar salvarea datelor ca fișiere CSV prin intermediul acestora se poate realiza foarte simplu.

Costuri și avantaje

***Care sunt costurile și avantajele unui asemenea tip de* e-business?**

Costurile de demarare și de întreținere necesare pentru o astfel de afacere *on-line* pot fi considerate ca fiind scăzute (cel puțin comparativ cu alte tipuri de afaceri), acestea ținând, în principal, de:

- înregistrarea unui nume de domeniu pentru situl Web;
- achiziționarea unui pachet de găzduire (*hosting*);
- proiectarea și implementarea sitului Web propriu-zis;
- întreținerea și îmbunătățirea permanentă a sitului;
- campaniile de promovare a sitului;
- oferirea de suport *on-line*, de preferat 24 de ore pe zi, 7 zile pe săptămână.

În afară de acestea, va mai fi necesară și angajarea unor persoane specializate în anumite domenii, fără de care această afacere nu și-ar putea desfășura activitatea, cum ar fi: avocați, juriști, agenți imobiliari, contabili, analiști informaticieni, specialiști în marketing.

În condițiile în care strategia de marketing este foarte bine pusă la punct, atunci și avantajele unei asemenea afaceri *on-line* vor fi pe măsura așteptărilor. Astfel, un prim avantaj (cel mai important, de altfel) este condiționat de realizarea unui număr cât mai mare de tranzacții imobiliare prin intermediul portalului, la fiecare tranzacție încheiată percepându-se un comision.

Creșterea numărului de tranzacții imobiliare încheiate se poate realiza prin oferirea unor prețuri promoționale pentru imobilele prezentate în cadrul portalului. Acest aspect va avea, de fapt, la bază cererea și obținerea de reduceri (*discount*-uri) de la agenții imobiliari care doresc să își prezinte ofertele în cadrul portalului. Dacă mai luăm în calcul și faptul că sumele vehiculate în cazul unor asemenea tranzacții imobiliare sunt destul de importante, atunci putem concluziona că și avantajele vor fi pe măsură.

De asemenea, mai există și alte avantaje în dezvoltarea unei afaceri *on-line* de acest gen, cum ar fi, spre exemplu, comercializarea de spații publicitare în cadrul portalului. Acest aspect va putea fi concretizat abia după ce situl va avea o popularitate destul de ridicată (număr mare de vizitatori pe zi, obținut prin consultarea rapoartelor de accesare a sitului).

Stabilirea identității sitului

Ce trebuie urmărit în proiectarea unei asemenea aplicații?

Alegerea unui nume pentru o aplicație, care să fie cât mai compatibil cu obiectivele acesteia, este un pas destul de important și, în același timp, dificil – trebuie avut în vedere faptul că acest nume formează prima impresie asupra aplicației. Pentru aceasta, se va alege o denumire de maxim două-trei cuvinte, întrucât un titlu nu poate fi prea lung. Expresia aleasă drept titlu trebuie să fie cât mai sugestivă, compatibilă cu scopul aplicației, și cât mai simplă, pentru a fi foarte ușor de memorat. De asemenea – de ce nu? – poate conține și un dram de ingeniozitate, pentru a se fixa mai bine în mintea utilizatorilor.

Astfel, având în vedere scopul portalului pe care intenționăm să îl dezvoltăm și principala activitate pentru care utilizatorii vor apela la serviciile oferite, am decis că numele „*FindMyHome*" se potrivește cel mai bine cu destinația aplicației. De asemenea, dacă se dorește conturarea mai bună a scopului aplicației, se poate recurge și la un *motto* care să aibă rolul de a întări mesajul exprimat de titlul ales.

Tot aici, în faza definirii unor elemente care să identifice (să reprezinte) aplicația, trebuie inclusă și conceperea unei sigle. Acest fapt presupune folosirea din plin a creativității, pentru a se realiza ceva cât mai deosebit, reușind ulterior să atragă ușor atenția asupra sa, oriunde va fi întâlnită sigla. Pentru cazul de față, am hotărât să alăturăm titlului *FindMyHome* imaginea unei căsuțe schițate în creion, ca și cum ar fi desenată de un copil.

De ce am insistat asupra acestor aspecte?

Rezultatele etapelor de definire a unui titlu (eventual a unui *motto*) și de concepere a siglei, care să fie cât mai compatibile cu aplicația ce urmează a fi dezvoltată, vor conduce la crearea unui simbol care va identifica (de dorit, în mod unic) aplicația oriunde va fi aceasta adusă în discuție (de exemplu, menționată în articole din ziare, în spații publicitare, în campaniile de promovare ș.a.m.d.).

Portalul nostru imobiliar are simbolul redat de următoarea imagine:

În campaniile de promovare se poate folosi și acesta:

care lasă impresia încrederii în încheierea unei afaceri.

Maniera de organizare a conținutului. Modalitățile de navigare

După cum am precizat în partea de început a acestui material, trebuie acordată foarte mare atenție modului în care este organizată și prezentată informația în cadrul sitului, manierei în care acesta este proiectat. Toate aceste detalii vor trebui să contribuie la o dirijare cât mai evidentă, facilă, a vizitatorului prin toate regiunile sitului și la o localizare rapidă a informației. Astfel, trebuie stabilite cu exactitate:

- modul în care situl va fi divizat și structurat în regiuni de interes;
- scopul fiecăreia dintre aceste regiuni;
- informația care urmează a fi folosită în cadrul acestora.

Un aspect important îl reprezintă maniera de organizare a meniului principal al aplicației. Vorbim de meniu principal întrucât, la un moment dat, într-o anumită regiune a sitului pot apărea, de asemenea, meniuri secundare,

al căror scop este de a structura local informația specifică acelei regiuni. Meniul principal trebuie să fie bine organizat, astfel încât să acopere tematicile importante ale sitului, și ar fi de preferat să nu conțină prea multe opțiuni, pentru a nu solicita utilizatorul cerându-i, indirect, să viziteze zonele mai puțin importante. Utilizatorul trebuie să-și păstreze modelul mintal al hărții navigabile a sitului, știind în orice moment ce opțiune trebuie să selecteze pentru a realiza o anumită acțiune. Astfel, denumirile opțiunilor stabilite trebuie să fie cât mai concise și să sugereze în mod corespunzător vizitatorului tipul informației grupate în cadrul unei opțiuni.

Pentru a nu supraîncărca meniul principal, dar și pentru a reliefa anumite facilități (de exemplu, serviciile oferite), accesul la acestea poate fi realizat din prima pagină prin intermediul unor zone bine definite, cum ar fi, de exemplu în cazul nostru, o zonă dedicată ofertelor speciale. De asemenea, unele servicii foarte folositoare, la care utilizatorii vor apela destul de des, pot fi integrate în acele părți ale sitului care vor apărea în toate paginile acestuia, cum ar fi *header*-ul sau *footer*-ul (este de preferat *header*-ul). Spre exemplu, în cazul unui portal, operațiunea de căutare este foarte necesară, drept pentru care am introdus în cadrul antetului un formular de căutare, accesibil din orice pagină a sitului.

În alegerea opțiunilor meniului principal mai trebuie luat în considerare și faptul că, printre opțiunile stabilite, vor trebui să fie incluse și unele a căror prezență a ajuns, la un moment dat, să facă parte dintr-un standard nescris al interfeței siturilor dedicate afacerilor *on-line*, cum ar fi:

- în primul rând, nu trebuie uitat faptul că prima opțiune, denumită sugestiv *Home* (*Acasă*), va reprezenta întotdeauna o legătură către prima pagină, acțiune care ar trebui să fie îndeplinită și de sigla sitului;
- nu trebuie să lipsească opțiunea *About Us* (în limba română *Despre noi*), în cadrul căreia se vor prezenta detalii privitoare la compania care desfășoară afacerea;
- va trebui inclusă și o secțiune conținând regulile de utilizare a sitului, cunoscută sub numele de *Terms and Conditions* sau *Privacy Policy*, cu echivalentul în limba română *Termeni și condiții*;
- în cazul unor situri de amploare, va fi binevenită o opțiune în cadrul căreia se vor prezenta și lămuri pentru posibilele probleme întâmpinate în utilizarea sitului, adoptându-se metoda întrebărilor și răspunsurilor (întrucât aceasta este cea mai ușor de înțeles); o astfel de secțiune e denumită prescurtat *FAQ* (*Frequently Asked Questions*) – *Întrebări frecvent formulate*;
- este de preferat să existe, de asemenea, o opțiune care să prevadă diverse posibilități de contact cu diferitele departamente ale firmei.

Căutarea informațiilor

În cadrul unui portal, un rol deosebit îl au metodele de căutare a informației cuprinse de acesta, întrucât acțiunea de căutare este una dintre funcționalitățile de bază ale acestui tip de sit Web. Astfel, pe lângă accesarea informației prin vizitarea diferitelor regiuni ale portalului (oferte speciale, anunțuri și altele), trebuie oferite vizitatorilor diverse posibilități de căutare avansată, particularizate în funcție de nevoile acestora. Este de preferat ca un astfel de instrument de căutare să fie prezent în orice regiune a portalului pentru a putea fi folosit imediat ce este nevoie, fără ca utilizatorul să părăsească pagina curentă ori să acceseze meniul de navigare.

Portalul imobiliar propus pune la dispoziția utilizatorilor două maniere de căutare a informațiilor. Una dintre ele este mai simplă, dar directă, oferind posibilitatea localizării unui imobil după numărul codului intern în cadrul portalului, cod denumit *FMH Number* (*FindMyHome Number*). Această modalitate de căutare este disponibilă din orice regiune a portalului, regăsindu-se peste tot în cadrul acestuia. Cea de-a doua soluție este mult mai complexă, fiind disponibilă direct din prima pagină a portalului și va fi prezentată pe larg în subcapitolul următor, dedicat analizei interfeței pentru clienți.

Modularizare și tehnologii folosite

Aplicația *FindMyHome* a fost proiectată în două module principale, fiecare dintre acestea având, la rândul lui, propria structură:

- prima parte este constituită din interfața pentru clienți, analizată în subcapitolul 4;
- a doua parte o reprezintă modulul de administrare a portalului, accesarea lui fiind restricționată vizitatorilor obișnuiți; acest modul oferă instrumentele necesare pentru întreținerea sitului.

La implementarea aplicației s-au folosit următoarele tehnologii:

- limbajul PHP, pentru generarea conținutului dinamic pe partea de server;
- limbajul JavaScript, pentru generarea conținutului dinamic pe partea de client și pentru validarea formularelor completate de utilizatori;
- limbajul de marcare HTML (*Hypertext Markup Language*), în ceea ce privește afișarea conținutului;
- limbajul CSS (*Cascading Style Sheets*), pentru formatarea conținutului;
- serverul de baze de date MySQL, pentru stocarea informațiilor;
- *cookie*-uri, pentru stocarea informațiilor pe partea de client.

În continuare, vom prezenta succint fiecare dintre cele două părți ale portalului dezvoltat.

4. Interfața pentru clienți

O afacere, în mod normal, trebuie gândită și desfășurată astfel încât să pună accent mai ales pe nevoile și cerințele clientului, întrucât acesta este cel care definește domeniul de activitate a acesteia. Astfel, și o afacere *on-line* trebuie organizată având la bază aceleași principii. O afacere *on-line* implică automat și existența unei aplicații software. Or, scopul unei asemenea aplicații este acela de a ajuta cât mai mult utilizatorul în activitățile sale, oferindu-i totodată și un cadru de lucru plăcut, relaxant, ușor de utilizat, scutindu-l de stresul și nervii metodelor clasice de rezolvare a anumitor probleme.

În desfășurarea unei afaceri nu trebuie uitate însă nici interesele celor care se ocupă de aceasta și o întrețin, chiar dacă ele nu vor fi prioritare față de cele ale clientului. Așadar, trebuie imaginate și dezvoltate modalități cât mai ușoare de întreținere și control (supraveghere) ale afacerii. Nu trebuie omis nici faptul că o afacere bine întreținută presupune efectuarea permanentă de modificări și îmbunătățiri pentru a se apropia cât mai mult de așteptările clienților. Aceste aspecte sunt importante cu atât mai mult în cazul desfășurării unei afaceri *on-line*.

Urmând precizările făcute anterior, aplicația *FindMyHome* a fost organizată astfel încât să fie pe cât mai mult posibil și ușor configurabilă.

La nivel de programare, aceste cerințe sunt îndeplinite prin folosirea fișierelor de configurare, conținând diverși parametri utilizați în cadrul aplicației (de exemplu, informațiile utilizate pentru conexiunea cu baza de date, pentru trimiterea de mesaje de *e-mail*, pentru specificarea structurii de directoare din cadrul aplicației și altele).

De asemenea, este pus la dispoziție un sistem oferind suport pentru exploatarea aplicației în mai multe limbi, inițial interfața fiind disponibilă în limba engleză. Astfel, în structura de directoare a aplicației este folosit un director pentru fiecare limbă în care se dorește să fie disponibilă aplicația. În acest director se includ informațiile dependente de limba folosită, cum ar fi: textul butoanelor din cadrul formularelor Web, imaginile din cadrul *design*-ului și constantele pentru mesajele afișate. Acestea vor fi încărcate în mod dinamic în momentul utilizării aplicației, în funcție de limba selectată la un moment dat.

La nivelul de design al aplicației, pe baza unor șabloane de afișare, au fost structurate zone al căror conținut este relativ independent și care poate fi deci facil de alterat sau chiar de înlocuit.

Aplicația *FindMyHome* este organizată inițial pentru a reprezenta piața imobiliară a orașului București, însă – conform celor prezentate anterior – aceasta poate fi modificată cu ușurință pentru a reprezenta piața imobiliară a oricărei alte localități și, chiar mai mult, a unui județ sau a unui stat. Ultimele două variante sunt posibile datorită implementării flexibile a motorului de căutare oferit.

Partea principală a aplicației *FindMyHome* o reprezintă motorul de căutare a locuințelor în baza de date.

Întrucât căutarea imobilelor va fi principala activitate pentru care vizitatorii vor apela la serviciile acestei aplicații, motorul principal de căutare a acesteia este inclus chiar în prima pagină a sitului, în cadrul căreia îi este rezervată o zonă importantă. Se poate astfel cu ușurință observa că accentul este pus pe serviciile oferite de către portal.

Pentru a evita confuziile, s-a adoptat ideea definirii vizuale a unor filtre mai generale, de alegere a zonei în care se dorește a fi amplasat imobilul căutat. Pentru a implementa această idee, trebuia folosită o imagine reprezentând harta orașului București.

Cum o asemenea hartă este destul de cuprinzătoare și greu de manipulat pe ecran, s-a recurs la metoda împărțirii hărții în mai multe regiuni și a structurării motorului de căutare pe mai multe niveluri. În cadrul fiecărui nivel, se definește câte un filtru vizual pentru restrângerea zonei în care se dorește a fi situat imobilul respectiv, utilizându-se harta regiunii selectate la nivelul anterior. Definirea unor asemenea filtre vizuale se realizează până când se ajunge la nivelul inferior, în care se cunoaște în detaliu zona unde trebuie situat imobilul căutat (aici vor fi definite efectiv filtrele de căutare privitoare la utilitățile clădirii și la alte informații).

În cazul nostru, primul nivel conține harta orașului București divizată în regiuni, după care, la următorul nivel, se folosesc hărțile regiunilor pentru a se defini zona de localizare a imobilului.

În situația reprezentării unui județ, primul nivel poate conține harta județului, din cadrul căreia se poate alege o anumită localitate, următoarele niveluri reducându-se la primul caz, de împărțire în regiuni a hărții localității respective. În cazul reprezentării unui stat, primul nivel poate conține harta acestuia, din cadrul căreia se poate alege un anumit județ, următoarele niveluri reducându-se la cazul reprezentării unui județ.

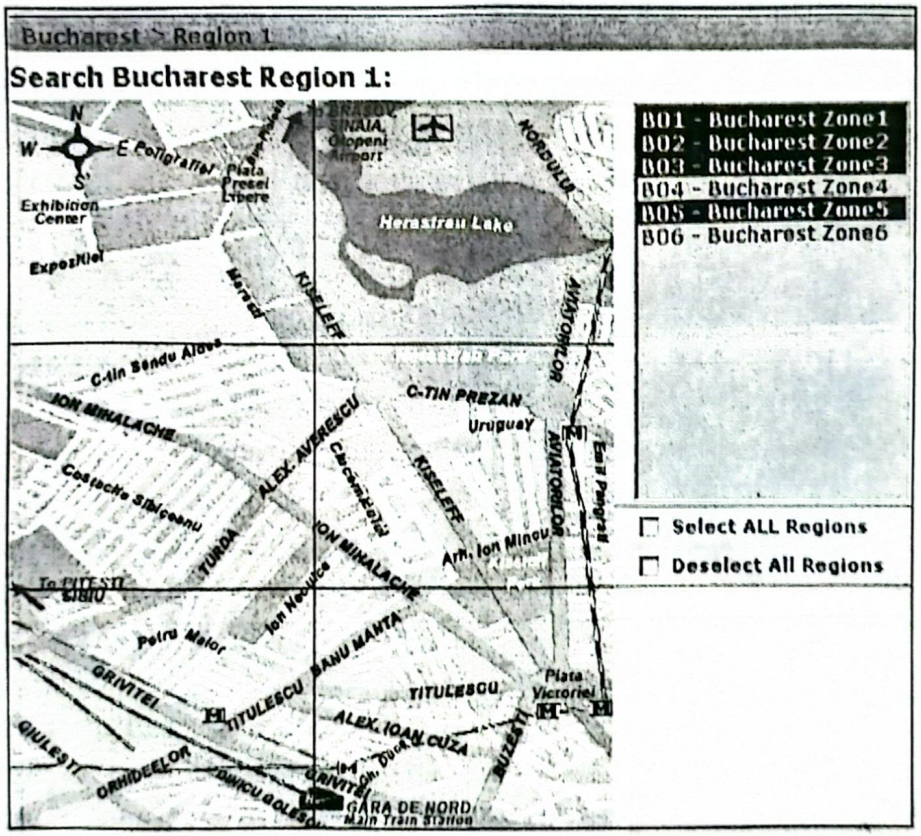
Dacă pentru o anumită regiune nu există în baza de date nici un imobil disponibil, aceasta nu mai poate fi selectată și nici detaliată (această situație este raportată, desigur, și vizitatorului).

Întrucât încărcarea imaginilor hărților poate necesita timp, s-a folosit un *preloader* implementat în JavaScript, utilizând DHTML (*Dynamic HTML*), care afișează o bară de progres al încărcării imaginii (*progress-bar*). Am adoptat această metodă pentru a nu afișa utilizatorului o pagină albă sau incompletă până la încărcarea integrală a imaginii. Acest aspect poate fi deranjant și chiar derutant pentru utilizator, putând sugera faptul că situl nu funcționează corect sau chiar deloc. Prima reacție a utilizatorului în acest caz va fi părăsirea sitului și formarea unei impresii greșite despre acesta, erodându-se imaginea portalului și, implicit, a companiei.

După alegerea zonei de localizare a imobilului căutat, utilizatorul poate să își filtreze căutarea conform următoarelor criterii:

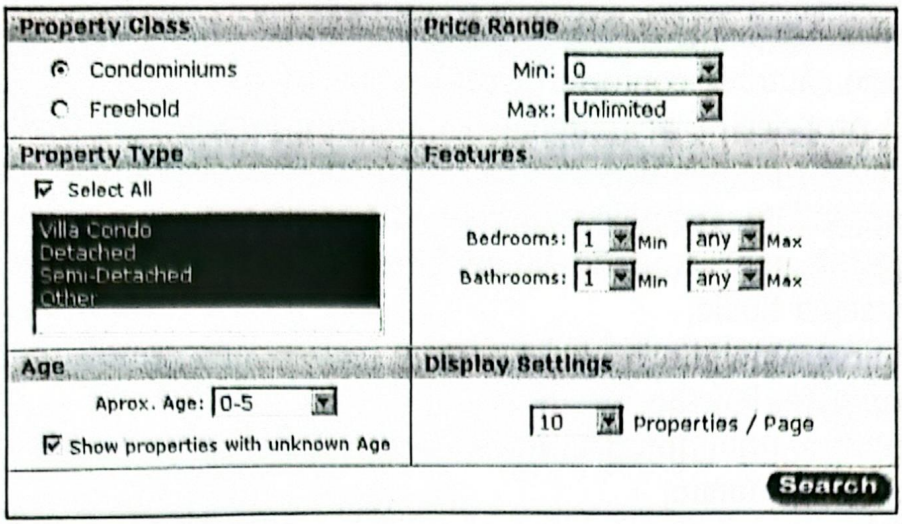
- tipul proprietății: apartament (*condominium*) sau casă (*freehold*);
- alegerea tipurilor de case sau apartamente disponibile, cum ar fi: decomandate, semidecomandate, vilă și altele;
- stabilirea unor anumite limite între care să se încadreze valoarea tranzacției finale;
- definirea unei limite inferioare și superioare pentru numărul de dormitoare, băi sau bucătării;
- selectarea unui interval între care trebuie să se încadreze vârsta imobilelor căutate;
- definirea numărului de proprietăți care vor fi listate pe o pagină.

Localizarea proprietății căutate se realizează prin alegerea uneia sau mai multor zone din cadrul regiunii curente, selectate la nivelul anterior, în care se preferă să fie situată aceasta, fapt ilustrat în următoarea figură:



Alegerea zonei/zonelor preferate

În funcție de imobilul selectat (apartament sau casă), filtrele de căutare referitoare la tipurile proprietăților și la utilitățile acestora diferă.



Criteriile de căutare a proprietăților

Rezultatele cu imobilele obținute în urma utilizării acestui motor de căutare sunt prezentate de următoarea captură de ecran:

Prezentarea rezultatelor căutării

După cum se observă, rezultatele imediate obținute în urma folosirii motorului de căutare conțin informații minimale, însă esențiale în procesul de comparare a proprietăților afișate și de realizare rapidă a unei impresii despre acestea.

Navigarea în cadrul rezultatelor furnizate se realizează prin intermediul unor butoane care facilitează defilarea prin paginile obținute. De asemenea, este oferit și un buton de schimbare a criteriilor de căutare, legătură care îl va redirecta pe utilizator la ultimul nivel de definire a criteriilor de căutare. În acest mod, vizitatorul nu este obligat să mai refacă toți pașii intermediari în specificarea criteriilor de căutare.

După cum se poate remarca în figura reprezentând rezultatele unei căutări, utilizatorul are posibilitatea de a se înscrie la o listă de *e-mail* pentru a fi notificat în privința rezultatelor obținute în urma filtrelor folosite. Astfel, acesta va fi ulterior contactat prin poșta electronică de fiecare dată când în cadrul portalului vor fi disponibile noi imobile care se potrivesc criteriilor de căutare alese.

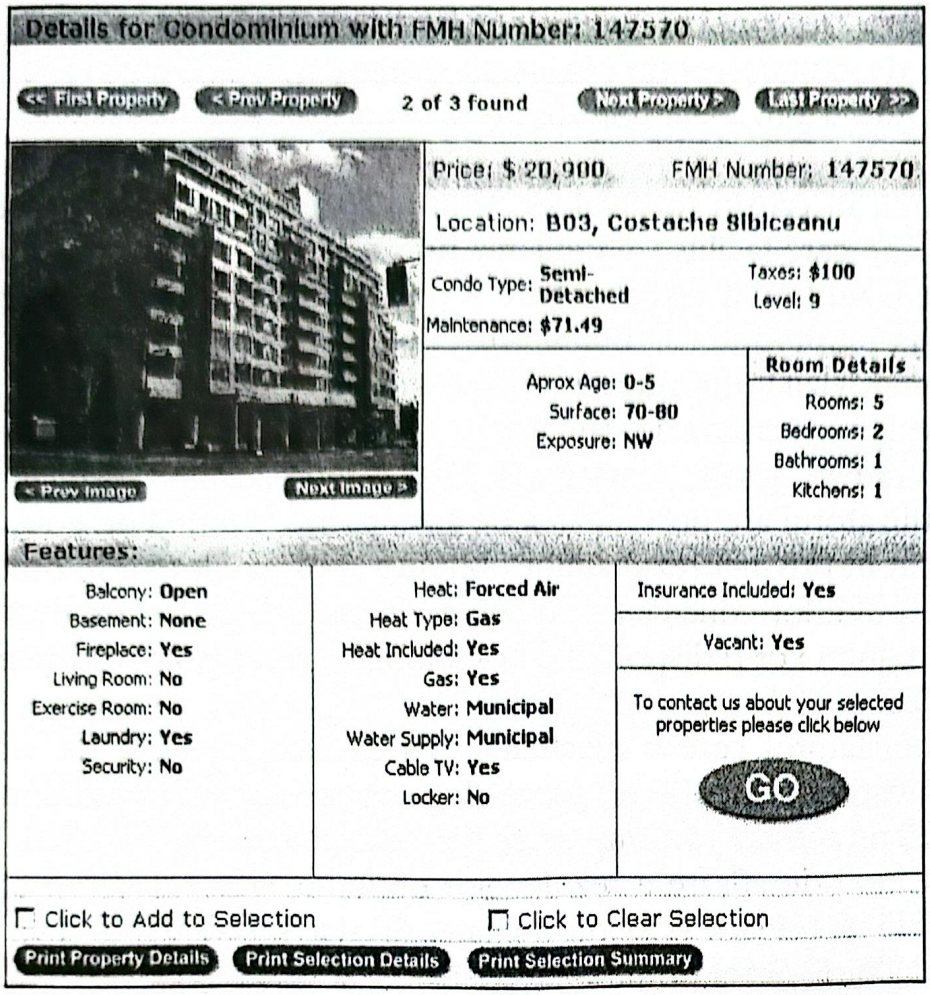
Dacă utilizatorul care a efectuat căutarea este interesat de un anumit imobil din cadrul celor afișate, poate alege opțiunea de a consulta mai multe detalii despre acesta. În urma selectării acestei opțiuni, se trece, de fapt, într-un alt mod de afișare a rezultatelor obținute, punând la dispoziție mai multe detalii privitoare la o proprietate. Și aici se oferă posibilitatea parcurgerii (defilării) listei de imobile găsite.

O opțiune destul de importantă a acestui modul de afișare este crearea de către utilizator a unei selecții cu imobilele care l-au interesat în mod deosebit. Această selecție poate fi consultată în orice moment și poate fi, de asemenea, listată la imprimantă, ca de altfel și detaliile despre o anumită proprietate. Maniera de listare la imprimantă este controlată cu ajutorul unor proprietăți CSS, astfel încât să fie listată doar informația necesară, excluzându-se elementele grafice de navigare prin sit (se utilizează `@media`).

Toate aceste aspecte cu privire la afișarea detaliată a informațiilor referitoare la imobilele obținute în urma căutării pot fi observate în următoarea captură de ecran.

Trebuie remarcat faptul că nu se pun la dispoziția utilizatorului informații cu privire la adresa exactă a imobilului, și nici modalități de contactare a proprietarului acestuia sau a agentului imobiliar care a furnizat informațiile despre imobil. Aceste măsuri de precauție au fost luate pentru a nu da posibilitatea utilizatorului de a lua legătura cu unul dintre aceștia, întrucât scopul aplicației este realizarea tranzacțiilor imobiliare prin intermediul sitului.

Astfel, dacă un utilizator este interesat de un anumit imobil, va trebui să completeze un formular de contact în care se precizează codul *FHM Number* al imobilului respectiv (acesta este completat automat de către sistem),



Prezentarea detaliilor despre imobile

datele personale, detaliile referitoare la modalitatea de contact şi un mesaj de intenţie. Ulterior, în funcţie de modalitatea de contact aleasă, utilizatorul va fi contactat de un reprezentant al sitului *FindMyHome* pentru a clarifica detaliile necesare încheierii unei posibile tranzacţii.

Trebuie menţionat faptul că implementarea unui sistem complet de comerţ electronic, presupunând şi efectuarea de plăţi *on-line* – prin utilizarea, de exemplu, a serviciilor oferite de *VeriSign* sau *Authorize.net* –, nu ar fi fost potrivită pentru această aplicaţie. În primul rând, efectuarea unor tranzacţii presupunând sume foarte mari trebuie realizată într-un mediu foarte sigur şi nu oricine ar avea încredere în realizarea unor astfel de plăţi. În al doilea rând, trebuie luată în considerare şi mentalitatea potenţialilor clienţi, care în nici un caz nu vor fi dispuşi să achiziţioneze un bun material de o asemenea valoare fără să aibă contact mai întâi cu acesta (un imobil, de cele mai multe ori, nu se cumpără frecvent într-o viaţă).

Cititorii interesaţi de interfaţa aplicaţiei *FindMyHome* au posibilitatea să testeze o variantă *on-line* la adresa: *http://www.webartwiz.com/fmh*.

5. Maniera de administrare a aplicației

Aplicația *FindMyHome* beneficiază, după cum am mai menționat, și de un modul de administrare, a cărui accesare este restricționată vizitatorilor neautorizați și care conține instrumentele necesare întreținerii portalului: managementul imobilelor, al agenților de vânzări, al listelor de ***e-mail*** din sistem, precum și managementul bazei de date.

În continuare, vom prezenta funcționalitatea fiecăreia dintre opțiunile de administrare a sistemului.

5.1 Managementul imobilelor

Pentru introducerea în baza de date a informațiilor privitoare la noile imobile, administratorul are la dispoziție o modalitate eficientă: importul de date din fișiere în format CSV. După cum am mai subliniat, folosirea acestui tip de fișiere asigură o portabilitate excelentă a datelor, acestea putând fi exportate/editate folosind orice program de calcul tabelar sau un editor de text obișnuit. Importul din fișiere CSV conduce la reducerea timpilor de administrare a bazei de date, fiind posibilă inserarea/modificarea rapidă a unui volum foarte mare de date.

Există două modalități de a importa fișiere CSV în cadrul aplicației noastre.

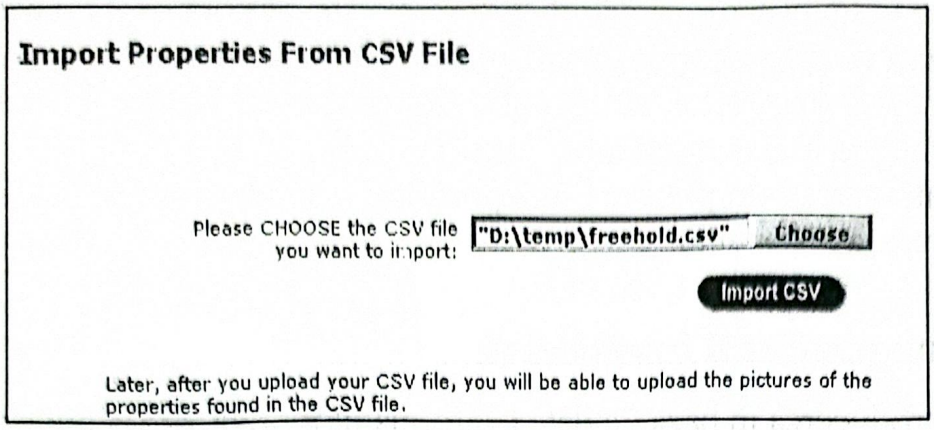
Prima metodă presupune *upload*-ul pe server a fișierului CSV via un formular Web. Imediat după *upload*, fișierul este procesat și sunt inserate sau actualizate în baza de date înregistrările valide din cadrul acestuia. În urma prelucrării conținutului fișierului, se detectează dacă este cazul să fie transferate și imaginile corespunzătoare imobilelor importate din înregistrările fișierului CSV. Dacă acest lucru este necesar, i se solicită în mod explicit utilizatorului, prin generarea unui formular de *upload* al respectivelor imagini.

A doua manieră presupune ca administratorul să aibă acces la un cont FTP (*File Transfer Protocol*) sau la un interpretor de comenzi (*shell*) pe sistemul pe care rulează aplicația. În acest caz, întreg procesul de importare a noilor informații este complet automatizat. Administratorul va trebui doar să copie în directorul *update* al aplicației fișierul sau fișierele CSV care conțin informațiile, cât și imaginile specificate pentru imobilele din cadrul fișierului/fișierelor respective. Următorul pas este cel de a selecta dintr-o listă, în cadrul unui formular Web, fișierul CSV care se dorește a fi importat. În urma acestei acțiuni, fișierul este procesat și sunt inserate sau actualizate automat în baza de date înregistrările valide din cadrul acestuia, și, dacă este cazul, se importă automat și imaginile pentru imobilele specificate.

La finalul acestui proces sunt afișate statistici cu privire la numărul imobilelor importate sau actualizate, precum și informații referitoare la imaginile transferate.

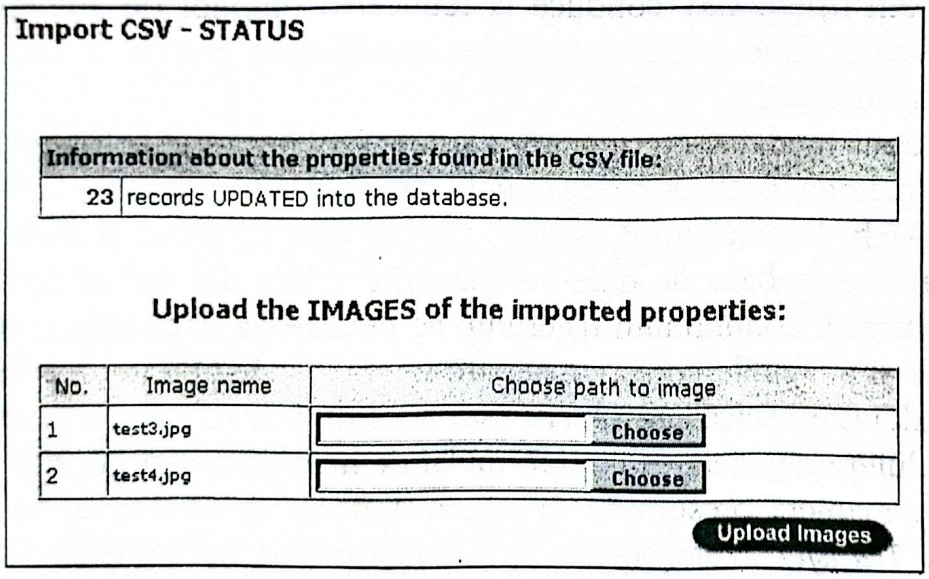
Prima metodă de import

Primul pas al acestei proceduri este *upload*-ul fișierului CSV dorit.



Transferul fișierelor în format CSV

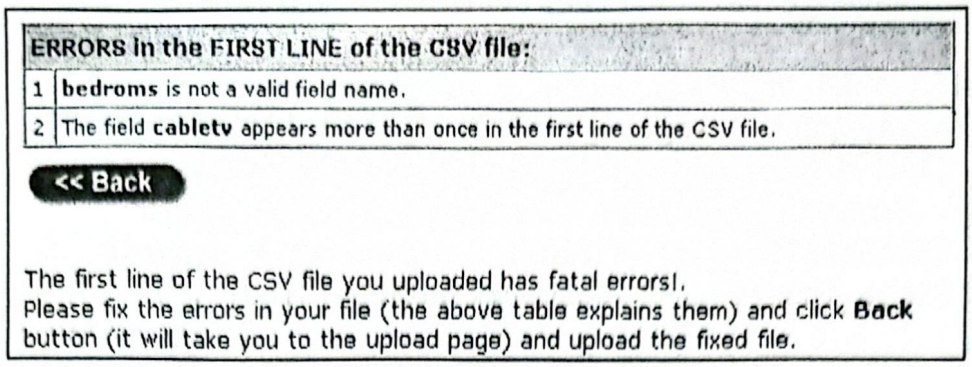
La următorul pas, sunt afișate statisticile referitoare la imobilele importate și este generat formularul de *upload* al imaginilor noi detectate, în cazul în care este necesar.



Statistici privitoare la datele importate și formularul pentru *upload*-ul imaginilor noi

Funcția de procesare a fișierelor CSV a fost implementată astfel încât să fie independentă de numărul câmpurilor și de pozițiile acestora în fișier. Consistența datelor se verifică prin compararea numelor coloanelor din prima linie (această linie este obligatorie și conține denumirile câmpurilor pentru informațiile din cadrul unei coloane – dacă nu există, se afișează mesajul de eroare corespunzător) cu elementele unui tablou (*array*) în care se precizează numele de câmpuri valide pe care trebuie să le conțină fișierul CSV.

În captura de ecran de mai jos se poate observa comportamentul aplicației în cazul procesării unui fișier CSV formatat eronat.



Procesarea unui fișier CSV eronat

Prezentăm codul sursă al funcției care se ocupă de procesarea fișierelor CSV – `parse_csv($cale_fisier)` –, lăsând cititorului misiunea de a reconstitui structura bazei de date folosite.

```
function parse_csv($cale_fisier) {
 // definim variabile globale
 global $user, $tabela_imobile, $imagini,
   $separator_default, $tabela_agenti, $campuri_admise,
   $erori, $inserate, $actualizate, $erori_header;

 // deschidem fisierul in format CSV pentru citire
 $fp = @fopen($cale_fisier,"r");
 // preluam prima linie
 $campuri_gasite = $pozitii = array();
 $avem_eroare = false;
 $into_insert = '';
 $prima_linie = fgetcsv($fp, 4096, $separator_default);
 $numar_campuri = count($prima_linie);

 // procesarea primei linii (header-ul)
 for($index=0; $index < $numar_campuri; $index++) {
   $prima_linie[$index] = strtolower($prima_linie[$index]);
   array_push($pozitii, 0);
   // numele campului nu este valid
   if(!in_array($prima_linie[$index], $campuri_admise)) {
     array_push($erori_header, "<b>$prima_linie[$index]</b>
       is not a valid field name.");
     $avem_eroare = true;
   }
   // numele campului este valid
   else {
     if(in_array($prima_linie[$index], $campuri_gasite)) {
       array_push($erori_header, "The field
```

```
            <b>$prima_linie[$index]</b> appears more than once
            in the first line of the CSV file." );
          $avem_eroare = true;
        }
        // e in regula
        array_push($campuri_gasite, $prima_linie[$index]);
        $ind = array_search($prima_linie[$index],
          $campuri_admise);
        $pozitii[$index] = $ind;
        $into_insert .= "`$campuri_admise[$ind]`,";
      }
    } // for - procesarea primei linii (header-ul)

    if($avem_eroare == true)
      return 0;

    // prelucram inregistrarile
    $into_insert = substr($into_insert, 0, -1);
    $linia = 1;
    // pozitiile unor campuri speciale
    $poz_price = array_search("price", $prima_linie);
    $poz_o_price = array_search("originalprice", $prima_linie);
    $poz_lastupdate = array_search("lastupdate", $prima_linie);
    $poz_imagini = array_search("images", $prima_linie);

    // sirul cu id-urile inserate/actualizate
    $ids = array();

    // procesam inregistrarile linie cu linie
    while($data = fgetcsv($fp, 4096, $separator_default)) {
      $insert = true;
      $linia++;
      $num = count($data);

      $values = '';
      $set_string = '';
      // avem acelasi numar de campuri ca in header?
      if($numar_campuri == $num) {
        // verificam daca avem in baza de date inregistrarea
        // cu mlsnumber curent, adica verificam daca exista
        // deja adaugat imobilul respectiv
        // mlsnumber = codul initial al imobilului, folosit in
        // cadrul sistemului agentului imobiliar
        $poz_mlsnum = array_search("origmlsnum", $prima_linie);
        if($poz_mlsnum !== null) {
          $bd = mysql_query("SELECT id, images
```

```
    FROM $tabela_imobile WHERE origmlsnum = '"
    addslashes($data[$poz_mlsnum])."'") or
      die(mysql_error());
$n = mysql_num_rows($bd);
// trebuie sa realizam o actualizare pt. acest imobil
if($n) {
  list($id_update, $images_update) =
    mysql_fetch_row($bd);
  $insert = false;
}
// trebuie sa facem o inserare
if($insert == true) {
  // construim sirul pentru VALUES() din query-ul SQL
  for($ii=0; $ii< $num; $ii++) {
    // pretul
    if($poz_price !== null && $ii == $poz_price) {
      $pretul = pret_csv2int($data[$ii]);
      $values .= "$pretul,";
    }
    else {
      // pretul initial
      if($poz_o_price !== null &&
         $ii == $poz_o_price) {
        $pretul_o = pret_csv2int($data[$ii]);
        $values .= "$pretul_o,";
      }
      else {
        // ultima actualizare
        if($poz_lastupdate !== null &&
           $ii == $poz_lastupdate) {
          $newdate = data_mysql($data[$ii]);
          $values .= "'$newdate',";
        }
        else {
          // imagini
          if($poz_imagini !== null &&
             $ii == $poz_imagini) {
            $imagini_curatate =
              parse_csv_images($data[$poz_imagini],
              "insert", '');
            $values .= "'$imagini_curatate',";
          }
          else
            $values .= "'" .
              addslashes(trim($data[$ii])) . "',";
        }
```

```
        }
      }
    } // final de for
    $values = substr($values, 0, -1);

    // introducem in baza de date
    mysql_query("INSERT INTO $tabela_imobile
      ($into_insert) VALUES($values)") or
        die(mysql_error());
    // aflam ID-ul inregistrarii
    $id_update = mysql_insert_id();
    $inserate++;
  }
  // trebuie sa facem o actualizare pentru imobilul
  // cu ID-ul $id_update
  else {
    for($ii = 0; $ii < $num; $ii++) {
      if($poz_price !== null && $ii == $poz_price) {
        // pretul
        $pretul = pret_csv2int($data[$ii]);
        $set_string .= "$prima_linie[$ii]=$pretul,";
      }
      else {
        // pretul initial
        if($poz_o_price !== null &&
            $ii == $poz_o_price) {
          $pretul_o = pret_csv2int($data[$ii]);
          $set_string .= "$prima_linie[$ii] =
              $pretul_o,";
        }
        else {
          // ultima actualizare
          if($poz_lastupdate !== null &&
              $ii == $poz_lastupdate) {
            $newdate = data_mysql($data[$ii]);
            $set_string .=
              "$prima_linie[$ii] = '$newdate',";
          }
          else {
            // imaginile
            if($poz_imagini !== null &&
                $ii == $poz_imagini) {
              $imagini_curatate =
                  parse_csv_images($data[$poz_imagini],
                    "update", $images_update);
              $set_string .=
```

```
                "$prima_linie[$ii]='$imagini_curatate',";
            }
            else
              $set_string .= "`$prima_linie[$ii]`='" .
                addslashes(trim($data[$ii])) . "',";
          }
        }
      }
    } // final de for
    $set_string = substr($set_string, 0, -1);
    mysql_query("UPDATE $tabela_imobile SET $set_string
      WHERE id=$id_update") or die(mysql_error());
    $actualizate++;
  }
} // finalul verificarii daca imobilul exista
  // in baza de date

// adaugam in sirul cu ID-urile inserate/actualizate
array_push($ids, $id_update);

// trimitem mail-uri de notificare
send_listing_mail($id_update);

// ne ocupam de detaliile despre agenti
$poz_nume_agent = array_search("agent", $prima_linie);
if($poz_nume_agent !== null) {
  $nume_agent = $data[$poz_nume_agent];
  // verificam daca agentul exista in baza de date
  $bdd = mysql_query("SELECT id FROM $tabela_agenti
     WHERE UPPER(agent)='" .
     addslashes($nume_agent)."'") or
       die(mysql_error());
  $este_ag = mysql_num_rows($bdd);
  // agentul exista: setam ID-ul agentului
  // pentru imobilul respectiv
  if($este_ag)
    list($id_ag) = mysql_fetch_row($bdd);
  // agentul nu exista: trebuie sa il adaugam
  else {
    $insert_what = "agent,";
    $insert_values = "'" .
       addslashes(strtoupper($nume_agent)) . "',";
    // extragem informatiile despre agent si le salvam
    $poz_agentphone = array_search("agentphone",
       $prima_linie);
    if($poz_agentphone !== null) {
      $insert_what .= "agentphone,";
```

```
            $insert_values .= "'" .
                addslashes(strtoupper($data[$poz_agentphone]))
                . "',";
            }
            $poz_agentoffice = array_search("agentoffice",
              $prima_linie);
            if($poz_agentoffice !== null) {
              $insert_what .= "agentoffice,";
              $insert_values .= "'" .
                addslashes(strtoupper($data[$poz_agentoffice]))
                . "',";
            }
            $poz_agentofficephonefax =
               array_search("agentofficephonefax",
                 $prima_linie);
            if($poz_agentofficephonefax !== null) {
              $insert_what .= "agentofficephonefax,";
              $insert_values .= "'" .
                addslashes(strtoupper($data[
                  $poz_agentofficephonefax])) . "',";
            }
            $insert_what .= "date_added, time_added";
            $insert_values .= "CURDATE(), CURTIME()";

            // adaugam agentul in baza de date
            mysql_query("INSERT INTO $tabela_agenti
              ($insert_what) VALUES($insert_values)")
              or die(mysql_error());
            $id_ag = mysql_insert_id();
          } // else - agentul nu exista in baza de date

          // setam codul agentului pentru acest imobil
          mysql_query("UPDATE $tabela_imobile
            SET agent_id=$id_ag WHERE id=$id_update") or
            die(mysql_error());
        } // if - exista campul agent

      } /* nu luam in considerare linia - nu are acelasi
           numar de campuri ca in antet */
      else
        array_push($erori, "LINE <b>$linia</b>:
          the number of fields is NOT EQUAL with
          the one in the first line.");
    } // final de while procesare inregistrari
```

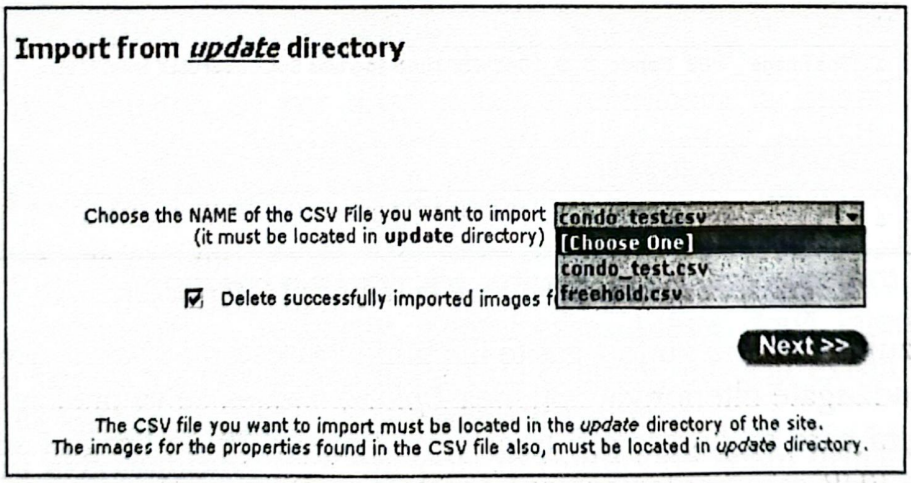
```
// inchidem fisierul CSV
fclose($fp);

// trimitem e-mail cu noile imobile adaugate utilizatorilor
// care s-au inscris la aceasta lista de email
if(count($ids))
  send_list_mail($ids);

return 1;
} // final de functie
```

A doua metodă de import

Această metodă presupune ca administratorul aplicației să aibă acces la un cont FTP sau SSH pe mașina pe care este găzduită aplicația.

În prima fază a procesului de import, administratorul trebuie să copie fișierul/fișierele CSV și imaginile imobilelor în directorul *update* al aplicației.

La următorul pas, în cadrul opțiunii *Import from dir* a modulului de administrare, se va selecta numele fișierului care se dorește a fi importat. Aceasta se realizează prin intermediul unui formular Web în care sunt incluse, sub forma unui câmp de intrare de tip *select*, toate fișierele CSV localizate în directorul *update* al aplicației, administratorul trebuind să aleagă unul dintre acestea. Importarea informațiilor se va realiza pentru fiecare fișier CSV în parte.



Selectarea fișierului care se dorește a fi importat

Porțiunea de cod care generează lista de tip *select* conținând fișierele CSV din directorul *update* (trebuie precizat faptul că în acest director se pot regăsi și fișiere în format imagine) este următoarea:

```
<select class="input_field" id="nume_fisier"
        name="nume_fisier" style="width:200px;">
       <option value="">[Choose One]</option>
```

```
        <?php
            if ($handle = opendir('../update/')) {
              while (false !== ($file = readdir($handle))) {
                 // ignoram fisierele speciale si pe cele care
                 // nu au extensia .csv
                if ($file != "." && $file != ".." &&
                      eregi("\.". 'csv'."$", $file) ) {
                   echo '<option value="' . $file . '">'
                      $file . '</option>';
                }
              }
              closedir($handle);
            }
        ?>
</select>
```

Fişierul ales va fi procesat, oferindu-se rezultatele acţiunilor efectuate:

Information about the properties found in the CSV file:	
11	records UPDATED into the database.
2	records INSERTED into the database.

IMPORT IMAGES STATUS:	
1	images SUCCESSFULLY IMPORTED.
1	images NOT FOUND in *update* directory.

IMPORT IMAGES DETAILS:	
1	The image **_W08_Condo_0_3_1048W391048.jpg** was SUCCESSFULLY IMPORTED.

IMPORT IMAGES ERRORS:	
1	The image **_W08_Condo_0_5_1403W418403.jpg** was NOT FOUND in *update directory*.

Afişarea datelor privitoare la importul informaţiilor

În cazul în care nu au fost găsite imagini în directorul *update*, acestea vor putea fi adăugate ulterior via opţiunea *Upload Images*. Se va preciza numărul de imagini care se doresc a fi transferate (maxim 20 de imagini simultan). Funcţiile PHP care implementează aceste operaţii sunt lăsate ca exerciţiu.

5.2 Modificarea detaliilor privitoare la un imobil

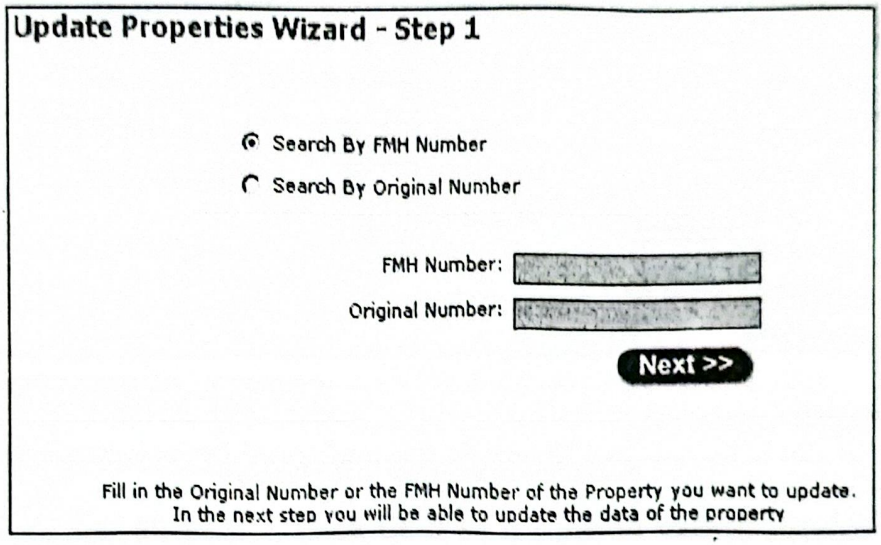
Pentru a actualiza detaliile unui imobil, administratorul are la dispoziţie două metode:

- folosirea facilităţii de import/actualizare din fişiere în format CSV, metodă care a fost descrisă anterior;

- editarea directă, prin intermediul unui formular Web, a informațiilor referitoare la acel imobil.

Pentru editarea directă, va trebui efectuată mai întâi o căutare a imobilului respectiv. Astfel, punem la dispoziția administratorului două metode de căutare:

- *Search By FMH Number* – presupune căutarea imobilului după codul intern al acestuia;
- *Search By Original Number* – oferă posibilitatea căutării după codul inițial al imobilului, folosit în cadrul sistemului agentului imobiliar care a furnizat informațiile referitoare la acest imobil.

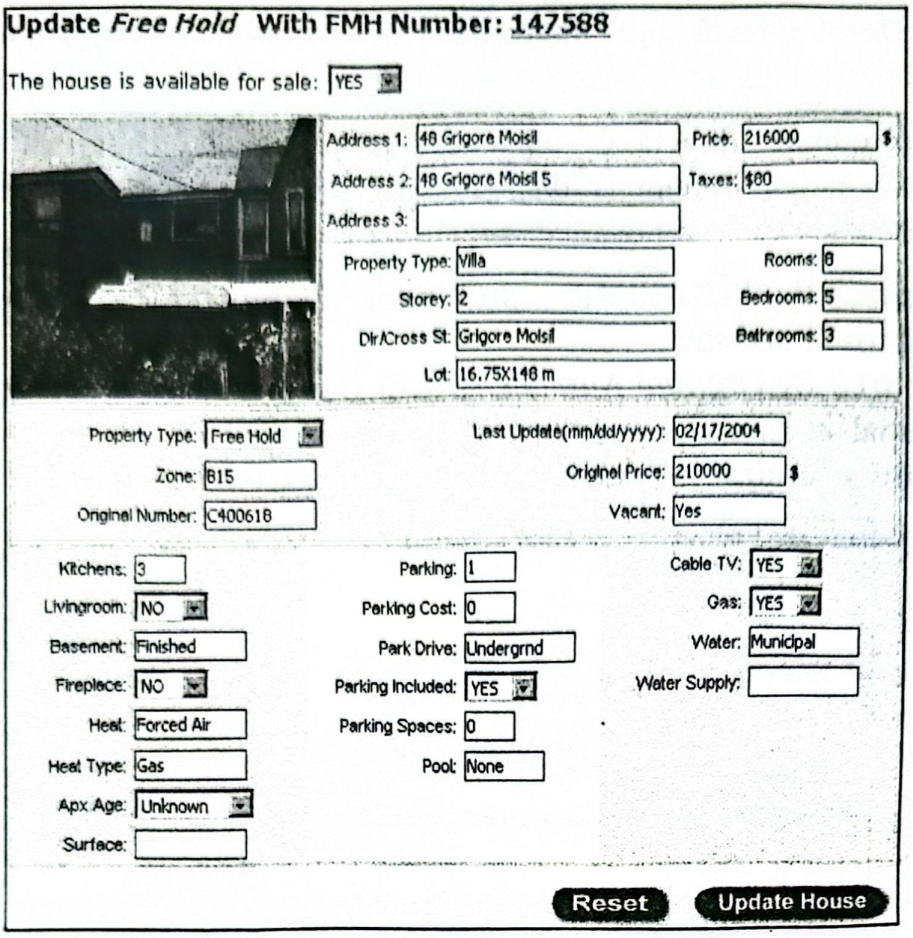


Căutarea unui imobil

La pasul următor, se vor afișa detaliile referitoare la imobilul căutat, oferindu-se posibilitatea modificării acestora, precum și posibilitatea de a specifica dacă acest imobil va fi sau nu afișat în căutările utilizatorilor acestei aplicații.

Există două interfețe distincte de editare a detaliilor pentru imobile: pentru apartamente și, respectiv, pentru case. Aceste două interfețe sunt necesare pentru a se evita producerea de confuzii, întrucât există diferențe în detaliile referitoare la cele două tipuri de imobile.

Pentru exemplificare, vom prezenta în captura de ecran următoare cum este concepută interfața de editare a detaliilor unei case.



Editarea detaliilor unui imobil

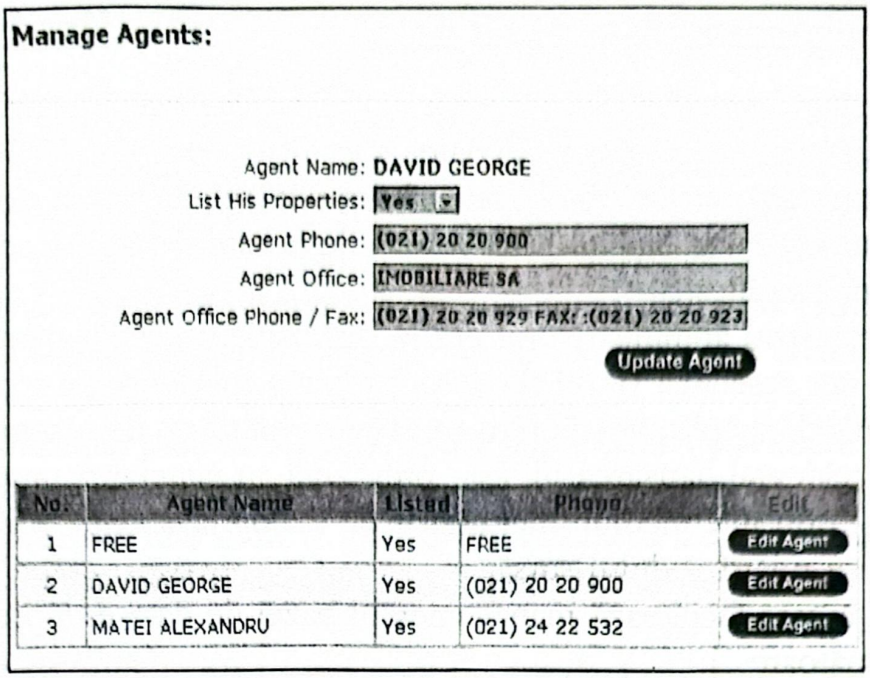
5.3 Managementul agenților imobiliari

La importul imobilelor din fișierele în format CSV, pot exista și câmpuri care să specifice detalii referitoare la agentul imobiliar care a furnizat informațiile privitoare la acele proprietăți.

Sistemul adaugă automat în baza de date agenții noi pe care îi identifică la efectuarea unui import al informațiilor dintr-un fișier CSV, activitate realizată în cadrul funcției de procesare a fișierului (prezentate în secțiunea 5.1 a capitolului de față).

De asemenea, mai sunt oferite și alte operații utile, cum ar fi posibilitatea editării informațiilor despre agenții imobiliari, precum și facilitatea de a specifica dacă proprietățile unui anumit agent vor fi sau nu afișate în căutările efectuate de utilizatori.

Interfața de administrare a agenților imobiliari este prezentată în următoarea captură de ecran:



Administrarea agenților imobiliari

Deoarece codul sursă al scripturilor de actualizare în baza de date a informațiilor preluate din aceste formulare este simplu, nu vom mai prezenta *listing*-ul acestora.

5.4 Managementul listelor de *e-mail*

Vizitatorii sitului au posibilitatea de a se înscrie în cadrul listelor de *e-mail* puse la dispoziție pentru a primi ulterior informații referitoare la noutățile din cadrul portalului.

Aplicația dispune de două liste:

- una generică, la care poate adera orice utilizator. Adresele de *e-mail* incluse aici vor fi folosite pentru a se expedia mesaje conținând noi oferte și servicii. Administratorul are posibilitatea de a trimite mesaje via modulul de administrare;
- o listă de *e-mail* la care utilizatorii se pot înscrie în urma efectuării unei căutări în cadrul aplicației. În momentul înscrierii, filtrele de căutare folosite vor fi memorate în scopul trimiterii prin poșta electronică a informațiilor referitoare la proprietățile noi adăugate în sistem, îndeplinind condițiile specificate de filtrele de căutare.

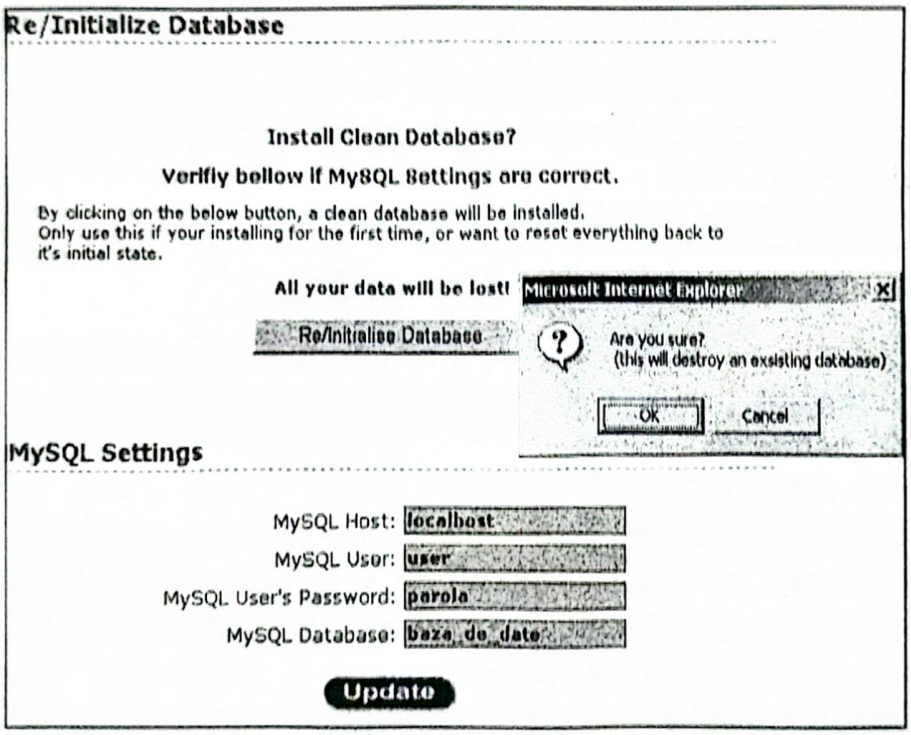
Din punctul de vedere al implementării, expedierea de mesaje prin poșta electronică este relativ simplă, fie folosindu-se funcția predefinită `mail()`, fie recurgându-se la comunicațiile cu un server SMTP (*Simple Mail Transfer Protocol*) prin intermediul funcțiilor dedicate programării în rețea (via *socket*-uri).

5.5 Managementul bazei de date

Administratorul are posibilitatea de a modifica direct configurările utilizate pentru conexiunea cu baza de date, specifice serverului MySQL folosit. Acest lucru este posibil deoarece informaţiile referitoare la configurările folosite sunt reţinute într-un fişier rescris de fiecare dată când aceste configurări sunt modificate.

De asemenea, administratorul are la dispoziţie un script pentru crearea tabelelor sau resetarea bazei de date, program folositor mai ales la configurarea iniţială a aplicaţiei. Pentru ca acest script să nu fie accesat şi utilizat în mod accidental, lansarea lui este prevăzută cu generarea unui mesaj de avertizare, prin intermediul unei căsuţe de dialog JavaScript, solicitându-se confirmarea rulării codului sursă.

Interfaţa care facilitează managementul bazei de date este prezentată în figura următoare:



Interfaţa de administrare a bazei de date

Acestea sunt, în mare, principalele opţiuni oferite de modulul de administrare a aplicaţiei *FindMyHome*, opţiuni menite să asiste întreţinerea şi supravegherea cât mai facilă a portalului imobiliar.

Recomandăm cititorilor interesaţi să implementeze în PHP sau în oricare alt limbaj de *server scripting* funcţionalităţile descrise mai sus sau să imagineze o altă soluţie de proiectare şi dezvoltare a sitului.

6. Concluzii

Prin maniera în care a fost structurată și implementată, putem concluziona că aplicația prezentată oferă un model de dezvoltare a afacerilor imobiliare *on-line*, reprezentând o posibilă soluție de succes. O alternativă ar fi aceea ca, în locul modulelor de administrare, să fie invocate servicii Web, iar interfața portalului să fie disponibilă pe mai multe platforme (calculatoare *desktop*, dispozitive mobile etc.), grație limbajului XML și a transformărilor XSL (*Extensible Stylesheet Language*).

Fără a furniza, în mod intenționat, foarte multe detalii privitoare la implementarea efectivă a scripturilor din care este compus portalul, capitolul și-a propus să prezinte mai mult o privire de ansamblu asupra unei idei de afacere *on-line*, autorul considerând că acest obiectiv a fost îndeplinit cu succes.

„Datoria" cititorului ar fi aceea ca, de fiecare dată când are o inițiativă de afacere *on-line*, să încerce să o analizeze atât din punctul de vedere al promotorului afacerii, cât mai ales din perspectiva clientului căruia i se adresează aceasta. Ar urma apoi, în cazul în care credeți că va avea șanse de succes, să încercați să o puneți în practică. În acest scop, o strategie de marketing foarte bine pusă la punct va fi de mare ajutor.

Pentru a putea face față eforturilor necesare în demararea unei astfel de afaceri, trebuie să avem încredere în posibilitățile proprii. De asemenea, trebuie să putem câștiga și încrederea unor terțe părți, care ar putea să ne ajute. Cu o documentare a proiectului foarte bine pusă la punct, se pot găsi investitori sau se poate chiar apela la un program de finanțare (internă sau externă).

De asemenea, nu trebuie uitat faptul că desfășurarea unei afaceri *on-line*, ca de altfel a oricărui alt tip de afacere, presupune asumarea unor riscuri și, din punctul de vedere al implementării finale, alegerea unei echipe profesioniste de designeri, programatori și administratori Web cu experiență.

7. Comentarii bibliografice

Domeniul deosebit de atractiv al *e-business*-ului poate fi aprofundat prin consultarea următoarelor cărți:

- D. Amor, *The E-Business (R)evolution*, Prentice-Hall, 1999;
- S. Buraga, *Proiectarea siturilor Web*, Polirom, Iași, 2002;
- B. Gates, *@faceri cu viteza gândului*, Amaltea, București, 2000;
- K. Laudon, C. Traver, *E-Commerce: Business, Technology, Society*, Addison-Wesley, 2003.

În ceea ce privește implementarea diverselor module ale portalului imobiliar descris în cadrul acestui capitol, recomandăm parcurgerea manualului PHP disponibil *on-line* la *http://www.php.net* – a se studia funcțiile `fgetcsv()`, `socket_()` și `mail()`. Alte volume de interes sunt B. Bulger, J. Greenspan, *MySQL/PHP Database Applications*, Willey, 2004 și L. Welling, *PHP and MySQL Web Development*, Sams, 2003.

O serie de soluții de implementare în PHP a aplicațiilor *e-business* sunt descrise în cartea *Aplicații Web la cheie*, coordonată de Sabin Buraga și apărută în anul 2003 la Editura Polirom – detalii la adresa *http://www.infoiasi.ro/~phpapps/*.

CAPITOLUL VII

Managementul *blog*-urilor în Perl

Sabin Buraga, busaco@infoiasi.ro

Acest capitol descrie fenomenul *Weblogging* şi maniera de implementare şi de exploatare a unui sistem rapid şi flexibil de *Weblog*-uri. Aplicaţia – rulând ca o suită de programe CGI – este scrisă în limbajul Perl. Funcţionalităţile de bază sunt implementate graţie expresiilor regulate şi modulului CGI.

1. Preliminarii

Capitolul de faţă oferă o serie de detalii privitoare la maniera de implementare în limbajul Perl a unui sistem de management al *Weblog*-urilor, sistem denumit *CSBlog*. Deşi există şi alte soluţii disponibile ca software liber sau comercial, aplicaţia prezentată în continuare reprezintă o infrastructură flexibilă şi uşor de extins, astfel încât să poată fi folosită cu succes atât în contextul *Weblogging*-lui, cât şi drept forum minimal.

Scopul principal al demarării unui astfel de proiect a fost acela de a demonstra că, folosind exclusiv instrumentele puse la dispoziţie de mediul Perl şi tehnologiile de marcare actuale (XHTML şi CSS), se poate concepe – în termen scurt – o aplicaţie Web utilă.

Fişierele cod sursă sunt scrise în limbajul Perl şi funcţionează ca programe CGI – *Common Gateway Interface* (serverul Web trebuie să ofere suport pentru CGI şi Perl). Această aplicaţie nu necesită suport pentru baze de date sau XML. Mediul Perl trebuie să pună la dispoziţie numai modulul CGI.pm (acesta este, în mod uzual, disponibil implicit la instalare).

Fiind software liber (*free*), *CSBlog* este disponibil la adresa *http://www.infoiasi.ro/~busaco/csblog/download/* (actualmente, există disponibile două versiuni, în limba engleză, respectiv în română). Doritorii pot redistribui şi/sau modifica aplicaţia, atât timp cât respectă termenii stipulaţi de licenţa *GNU General Public License* (a se vedea *http://www.fsf.org/licenses/*).

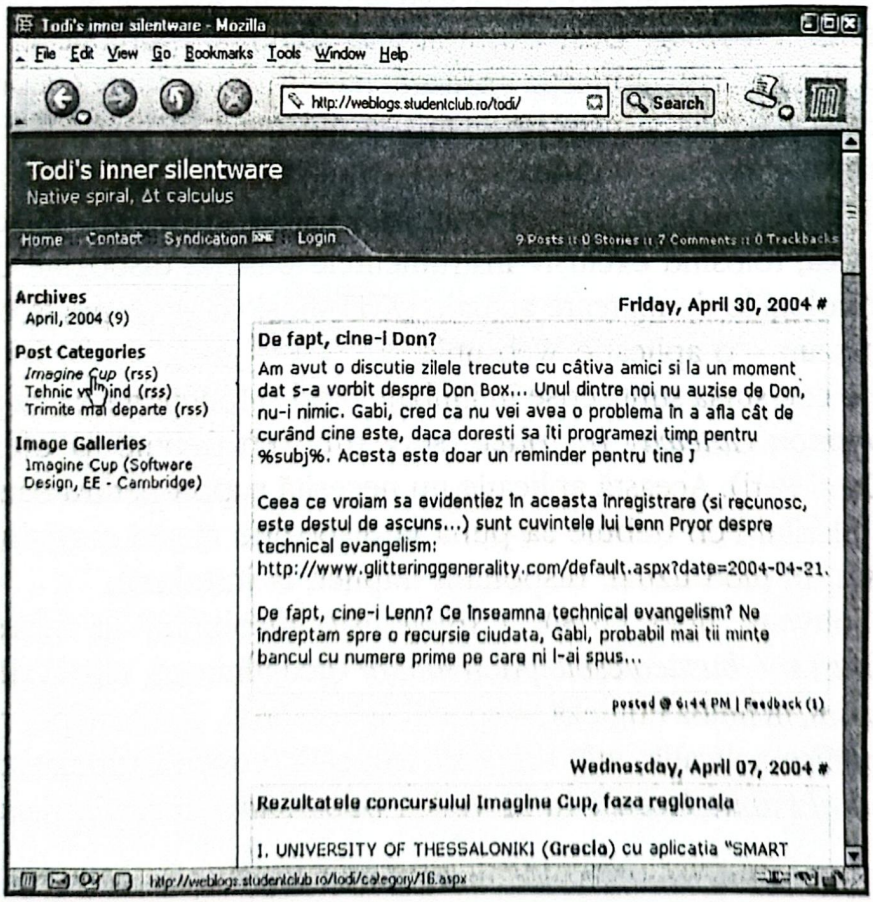
2. Ce este un *Weblog*?

Un *Weblog* (sau *blog*, pe scurt) reprezintă o pagină Web care conține diverse fragmente de informații puse la dispoziție de o persoană. Mai mult decât atât, un *blog* poate fi considerat un mijloc de comunicare interpersonală, natura sa putând varia de la o formă de jurnal *on-line*, până la un depozit de informații privitoare la un domeniu (tehnic sau nu), fiind editat fie de o singură persoană, fie de un grup.

Fenomenul *Weblog* a început să ia amploare din anul 2000, unii analiști estimând că zilnic apar pe Web peste 80 de mii de noi *blog*-uri, întreținute atât de anonimi, cât și de diverse personalități provenind dintr-o sumedenie de domenii.

Anatomia unui *blog* este simplă, structura sa incluzând un titlu (eventual un subtitlu), o listă a mesajelor (intrărilor) curente, redactate de autorul/autorii *blog*-ului ori de terți vizitatori, o listă de legături spre alte situri sau *blog*-uri de interes (așa-numitul *blogroll*) și, eventual, diverse alte ingrediente (sigla, liste de discuții, statistici etc.).

Unele subiecte interesante pot fi discutate separat, utilizatorii având posibilitatea, via unor formulare Web, să introducă propriile opinii sau sugestii în cadrul *blog*-ului. Desigur, discuțiile pot fi monitorizate de proprietarul *blog*-ului, iar unele mesaje necorespunzătoare pot fi eliminate complet sau parțial.



Exemplu de *blog* personal

Pentru a putea referi un anumit mesaj (care poate dispărea de pe prima pagină a *blog*-ului dacă dinamica sa este mare), utilizatorii pot să se folosească de așa-numita legătură permanentă (*permanent link*) asociată acestuia. Astfel, un mesaj poate fi accesat fie direct, fie parcurgând lista mesajelor (*post*-urilor) existente.

O componentă opțională o reprezintă motorul de căutare internă, vizitatorii având posibilitatea să regăsească, pe baza unor criterii (de exemplu, cuvinte-cheie incluse în subiect, data redactării, autor etc.), informațiile cuprinse în lista mesajelor *blog*-urilor.

Ca exemple de unelte de administrare a *Weblog*-urilor pot fi menționate:

- *Blogger – http://www.blogger.com*
- *Blosxom – http://www.raelity.org/lang/perl/blosxom*
- *LiveJournal – http://www.livejournal.com*
- *Manila – http://manila.userland.com*
- *MovableType – http://www.movabletype.org*

Există, desigur, o multitudine de situri de găzduire (*hosting*) a *blog*-urilor. Unele oferă găzduire gratuită și diverse instrumente simple de administrare (*Blogger* sau *LiveJournal* – la nivel internațional – sau *WeBlog*, disponibil la *www.WeBlog.ro* – pe plan național). O altă soluție este instalarea pe propriul server Web a aplicației de management al *blog*-ului, cele mai multe programe rulând ca scripturi CGI (de exemplu, *Blosxom*, fiind scris în Perl).

De asemenea, cele mai multe *Weblog*-uri dau posibilitatea programelor (roboți sau agenți Web) să indexeze ori să manipuleze conținutul mesajelor via fișierele *RSS* (*Rich Site Summary*) asociate. Formatul RSS permite calculatoarelor să interschimbe informații privitoare la conținutul siturilor Web (în acest caz, al *blog*-urilor). Fiind bazat de XML, un document RSS este ușor de procesat sau de creat, existând diverse instrumente – numite *agregatori de conținut RSS* – care extrag, sumarizează și stochează datele RSS. Ca exemple de agregatori RSS putem menționa *Syndic8 – www.syndic8.com* – sau *NewsIsFree – www.newsisfree.com*. Există și posibilitatea utilizării unor agregatori personali, ca *AmphetaDesk – www.amphetadesk.com*.

Unele sisteme oferă și posibilitatea procesării *blog*-urilor via o interfață de programare, denumită *Blogger API*. Aceasta folosește protocolul XML-RPC, astfel încât managementul mesajelor se poate realiza de la distanță, folosind – de exemplu – un client Windows special. Alte soluții de găzduire expun o suită de servicii Web administrative care pot fi invocate prin SOAP (*Simple Object Access Protocol*).

Există și aplicații care integrează un sistem de *blogging* în cadrul unui portal Web. Un exemplu notabil este *Slash – www.slashcode.org* –, disponibil gratuit ca aplicație Web scrisă în Perl.

3. Implementarea aplicației *CSBlog*

După ce am văzut care este structura unui *Weblog*, în cadrul acestui subcapitol vom descrie modul de implementare a propriei noastre soluții. Aplicația a fost dezvoltată, testată și exploatată sub mediul Perl din distribuțiile Linux actuale. Ea poate rula și sub Windows, dacă a fost instalat unul dintre pachetele gratuite *ActivePerl* sau *Apache2Triad*.

3.1 Modularizarea. Fișierele constituiente

Vom modulariza codul astfel încât să oferim o flexibilitate ridicată. De exemplu, utilizatorii vor dori să-și modifice preferințele în ceea ce privește interfața, iar programatorii vor intenționa să extindă *CSBlog* prin includerea unor noi funcționalități, precum atașarea unui calendar, a unui album de fotografii sau a unei liste de noutăți preluate de pe alte situri via RSS.

În varianta actuală, fișierele care compun aplicația *CSBlog* sunt grupate astfel (conform ieșirii comenzii `tree` din UNIX/Linux):

```
/
|-- History     istoricul versiunilor
|-- Readme      informatii privitoare la aplicatie
|-- config.pl   include variabilele de configurare
|-- core.pl     reprezinta nucleul CSBlog
|-- csblog.cgi  CGI care afiseaza continutul blog-urilor
|-- csblogs     director care stocheaza mesajele
|   |-- Download
|   |   `-- Sun%20Aug%2015%2013%3A40%3A00%202004.csblog
|   `-- Help
|       |-- Mon%20Aug%20%209%2014%3A40%3A06%202004.csblog
|       |-- Mon%20Aug%20%209%2015%3A10%3A12%202004.csblog
|       |-- Mon%20Aug%20%209%2015%3A44%3A47%202004.csblog
|       `-- Tue%20Aug%2010%2015%3A34%3A18%202004.csblog
|-- enblog.cgi  CGI care afiseaza lista intrarilor
|-- index.cgi   CGI principal; listeaza blog-urile curente
|-- pics        ,director continand pictograme - smileys
|   |-- icon_biggrin.gif
|   |-- icon_confused.gif
|   |-- icon_cool.gif
|   |-- icon_exclaim.gif
|   |-- icon_idea.gif
|   |-- icon_lol.gif
|   |-- icon_mad.gif
|   |-- icon_neutral.gif
|   |-- icon_question.gif
|   |-- icon_razz.gif
|   |-- icon_sad.gif
```

```
|    |-- icon_smile.gif
|    |-- icon_surprised.gif
|    `-- icon_wink.gif
|-- preferences.cgi CGI care modifica preferintele
|-- styles      director stocand temele vizuale - fisiere CSS
|    |-- _csblog.css
|    |-- blues.css
|    |-- endirrilium.css
|    |-- ethereal.css
|    |-- lemon.css
|    `-- sunny.css
|-- templates  director ce include sabloanele de interfata
|    |-- footer.tmpl
|    |-- form.tmpl
|    |-- header.tmpl
|    |-- nbform.tmpl
|    `-- upform.tmpl
`-- utils.pl    diverse functii utile
```

După cum se poate remarca, programul CGI principal este *index.cgi*, care oferă lista *blog*-urilor existente (sunt disponibile două *blog*-uri cu informații privitoare la ajutor și la disponibilitatea surselor).

Alt program CGI este *csblog.cgi*, invocat pentru vizualizarea detaliilor privitoare la *blog*-uri și parcurgerea conținutului acestora. Utilizatorii vor putea adăuga propriile mesaje, recurgând la diferite facilități oferite (inserarea automată a hiperlegăturilor, emoticonuri (*smileys*), suport pentru unele *tag*-uri speciale etc.). Un *blog* poate fi închis (nu se mai pot adăuga alte mesaje, deci formularul de introducere a unui alt *post* nu mai este afișat) sau nu.

Lista intrărilor (mesajelor sau *post*-urilor) unui *blog* este generată de scriptul *enblog.cgi*. Fiecare intrare are atașată o hiperlegătură (de fapt, o legătură permanentă) spre mesajul complet care va fi afișat de *csblog.cgi*.

Pentru a schimba diferite preferințe ale utilizatorului, se poate invoca scriptul *preferences.cgi*. Vizitatorii își pot stabili identitatea și afilierea (URI-ul personal, precum și adresa de *e-mail* proprie sau cea a sitului Web) și pot modifica tema vizuală a interfeței *CSBlog* – sunt disponibile mai multe teme, reprezentate de fișiere CSS stocate în directorul *styles/*. Preferințele sunt memorate de un *cookie* setat ca permanent.

Toate functiile de „nivel scăzut" sunt încapsulate de fișierul *core.pl*. Alte fișiere Perl sunt *config.pl* (folosit la configurarea *CSBlog*-ului) și *utils.pl* (care definește anumite rutine utile, precum lucrul cu șabloane de afișare sau afișarea datei).

Fișierele suplimentare (ale căror nume sunt terminate cu extensia *.tmpl*) reprezintă diferite fragmente de marcaje XHTML pentru crearea interfețelor Web, localizate în directorul *templates/*. Definițiile generale de stiluri (fișierele de teme CSS – nivelul 2) sunt stocate în directorul *styles/*. Emoticonurile

(*smileys*) afișate în cadrul mesajelor și inserate de utilizatori prin diferite simboluri speciale sunt fișiere GIF preluate de la forumul *phpBB*, imagini localizate în directorul *pics/*.

Conținutul propriu-zis al mesajelor introduse în *blog*-uri este memorat în directorul *csblogs* (localizarea acestuia poate fi modificată, prin intermediul unei variabile setate în fișierul *config.pl*). Pentru fiecare *blog*, va fi creat un director în cadrul directorului *csblogs* (după cum am afirmat, există deja două directoare – *Help* și *Download* – care stochează *blog*-urile, conținând informații de ajutor și referitoare la posibilitățile de *download* al aplicației). Fiecare mesaj al unui *blog* reprezintă, de fapt, un fișier XHTML, extensia sa implicită fiind *.csblog*. Numele său este o codificare specială a timpului de creare a mesajului:

Ajustarea interfeței se poate realiza prin editarea fișierelor *header.tmpl*, *footer.tmpl* și *styles/*.css*. Formularul de editare a mesajelor este stocat de fișierul *form.tmpl*, iar formularul utilizat la crearea unui *blog* este disponibil în *nbform.tmpl*. Formularul de editare a preferințelor utilizatorilor este memorat de fișierul *upform.tmpl*.

3.2 Variabilele globale. Fișierul de configurare

Pentru a înțelege modul de implementare a funcționalităților de bază, trebuie, în primul rând, să vedem ce anume poate configura utilizatorul (în acest context, administratorul aplicației). Fișierul *config.pl* conține declarații de variabile globale ale căror valori pot fi modificate de către utilizator. Astfel, dăm posibilitatea ajustării unor parametri privitori la crearea de noi *blog*-uri, reținerea adresei clientului, stabilirea numelui utilizatorului și afilierii lui etc. Tot aici se pot schimba numele directoarelor și fișierelor care stochează șabloanele de afișare sau stilurile. De asemenea, se pot stabili – via variabila `$blogs` – numele și locația directorului care va memora mesajele *blog*-urilor.

Numele *cookie*-ului care va reține preferințele utilizatorilor este stocat în variabila `$cookie_name`, iar durata sa de viață în variabila `$cookie_life`.

3.3 Crearea interfeței și stabilirea preferințelor

Pentru a genera o interfață flexibilă, vom utiliza diverse șabloane (*templates*) de prezentare. Soluția este simplă: fișierele XHTML vor putea conține diferite referiri la variabile Perl care, prin intermediul unei funcții, vor fi substituite cu valorile lor actuale. Astfel, în antetul fiecărei pagini vom putea indica numele *blog*-ului curent, iar în subsol vom include legături către situl *CSBlog* și spre autorul acestuia ori vom oferi posibilitatea modificării preferințelor.

De exemplu, antetul fiecărei pagini generate dinamic va putea conține următoarele marcaje XHTML:

```
<!DOCTYPE html
   PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN".
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<?xml-stylesheet href="<!!styles!!>" type="text/css"?>
<html xmlns="http://www.w3.org/1999/xhtml" lang="ro">
<head>
<title>:: CSBlogs :: <!!topic!!></title>
<meta http-equiv="Content-Type"
content="text/html; charset=ISO-8859-2" />
<link rel="stylesheet" href="<!!styles!!>"
    type="text/css" />
<!-- metadate -->
<meta name="description" content=
"CSBlog :: personal blog of Sabin-Corneliu Buraga" />
<meta name="keywords" content=
"weblog, blog, <!!topic!!>" />
<meta name="author" content=
"Sabin Corneliu Buraga - busaco@infoiasi.ro" />
<!-- evitam 'image toolbar' din IE 6 -->
<meta http-equiv="imagetoolbar" content="no" />
</head>
<body text="#000000" bgcolor="#FFFFFF">
<a name="top" id="top"></a>

<div class="table">
```

Acest antet este stocat de *header.tmpl*. Finalul de pagină va fi memorat în fișierul *footer.tmpl* și va putea avea următorul conținut:

```
<hr size="1" />
<div class="time author">
<a href="preferences.cgi" title="Schimbati preferintele"
accesskey="P">Preferinte</a> ·
<!!csblog_version!!> · <a href="#top"
title="La inceputul acestui document"
accesskey="S">Sus</a></div>
</div>
</body>
</html>
```

Am folosit atributul `accesskey` pentru ca navigarea prin sit să se poată efectua prin intermediul tastaturii. Acest aspect mărește gradul de accesibilitate al sitului, mai ales pentru persoanele cu diverse handicapuri locomotorii – a se consulta și documentele referitoare la WAI (*Web Accessibility Initiative*) de pe situl Consorțiului Web.

Funcția care realizează substituția valorilor variabilelor, adică generează ieșirea XHTML pornind de la fișierul-șablon, este următoarea (codul sursă este localizat în fișierul *utils.pl*):

```
# genereaza (X)HTML pornind de la un sablon,
# inlocuind "<!! ... !!>" cu valoarea unei variabile
sub gen_output() {
  my $template = shift;
  %vars = @_;
  my $output; # retine codul generat

  return unless open (TEMPLATE, "<$template");
    # nu putem deschide fisierul

  # citim continutul fisierului
  while (<TEMPLATE>) {
    $output .= $_;
  }
  close (TEMPLATE);

  local(@keys, $key, $params, @param);

  # determinam ce trebuie sa substituim
  @keys = ($output =~ /<!!(.*?)!!>/g);

  foreach $key (@keys) {
    if (defined($$key)) {
      $output =~ s/<!!$key!!>/$$key/g;
    }
    elsif (defined($vars{$key})) {
      # inlocuim cu valoarea variabilei
      $output =~ s/<!!$key!!>/$vars{"$key"}/g;
    }
    else {
      # n-am gasit, generam entitatea ' '
      $output =~ s/<!!$key!!>/\ /g;
    }
  }
  print $output;
  return 0;
}
```

Recurgând la această abordare, putem schimba foarte ușor întreaga interfață sau putem genera conținutul unor porțiuni din documentul rezultat. Cu ajutorul șabloanelor, reușim să ajustăm stilurile în funcție de preferințele stabilite de utilizator. De fapt, în antetul paginii amplasăm construcția `<!!styles!!>` pentru a putea fi substituită cu numele fișierului CSS preluat din *cookie*-ul de pe calculatorul clientului.

O alternativă la abordarea Perl descrisă mai sus este *PHP Smarty*, disponibil la adresa *http://smarty.php.net/*.

Scriptul de modificare a preferințelor utilizatorului va seta un *cookie* persistent care va memora numele fișierului de stil (tema vizuală), plus

numele utilizatorului şi afilierea sa, spre a fi introduse automat în formularul de editare a unui mesaj.

Codul sursă al acestui program CGI este stocat de *preferences.css*:

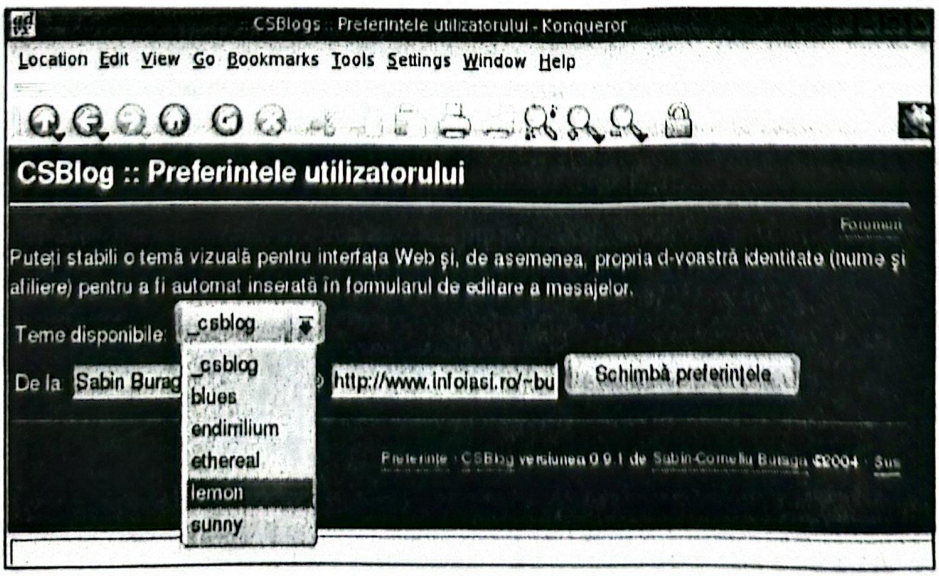
```
#!/usr/bin/perl
# seteaza preferintele-utilizator

require "core.pl";

# preia vechile preferinte din cookie (daca exista)
%prefer = CGI::cookie($cookie_name);

# preia diverse cimpuri via variabilele HTTP
$styles = CGI::param('styles')
   || $prefer{'styles'} || $styles;
$name = CGI::param('name')
   || $prefer{'name'} || $default_name;
$uri = CGI::param('uri')
   || $prefer{'uri'} || $default_uri;

# verificam daca au fost setate
$prefer{'styles'} = $styles if $styles;
$prefer{'name'} = $name if $name;
$prefer{'uri'} = $uri if $uri;

# cream cookie-ul persistent
# (maxim $cookie_life zile) si-l trimitem
# inainte de a genera continutul XHTML
$cookie = CGI::cookie(
   -name => $cookie_name,
   -value => \%prefer,
   -expires => '+' . $cookie_life . 'd');
print CGI::header(-cookie => $cookie);

# pseudo-subiect
$topic = 'Preferintele utilizatorului';
# generam antetul
&gen_output($header);

# generam continutul paginii
print "<h2 class=\"header\">$default_heading ::
<span class=\"subject\">$topic</span></h2>";

# inseram structura navigationala
print "<div class=\"time\"><a href=\"index.cgi\"
title=\"Afiseaza forumurile disponibile\"
accesskey=\"F\">Forumuri</a></div>\n";
```

```
# generam formularul Web
&gen_prefer;

# generam subsolul paginii
&gen_output($footer);
```

Funcţia `gen_output()` a fost descrisă mai sus, variabilele `$header` şi `$footer` conţinând numele fişierelor-şablon folosite. Funcţia `gen_prefer()` generează lista temelor vizuale existente şi apoi creează formularul Web de editare a preferinţelor:

```
sub gen_prefer() {
  # genereaza lista stilurilor disponibile
  # si o include intr-un marcator "select"
  @styles = glob("$styles_dir/*.css");
  $styles = "<select name=\"styles\"
    title=\"Apasati aici pentru a alege tema
    favorita\">\n";
  foreach $styles_item (@styles) {
    $_ = $styles_item;
    # ignoram ce nu se termina cu '.css'
    s{(.+\/)(.+)\.css} {$2}gs;
    $styles_name = $_;
    $styles .= "<option value=\"$styles_item\">
      $styles_name</option>\n";
  }
  $styles .= "</select>\n";

  &gen_output($upform);
}
```

Formularul Web este ilustrat în figura următoare:



Stabilirea preferinţelor utilizatorului

3.4 Afișarea *blog*-urilor existente

După cum menționam mai sus, programul principal va afișa lista *Weblog*-urilor existente și va oferi legături spre conținutul fiecărui *blog* în parte. De asemenea, dacă variabila `$create_blogs` are o valoare nenulă, atunci va da vizitatorilor posibilitatea creării unui alt *blog*. Scriptul este numit *index.cgi*, codul lui fiind foarte simplu:

```
#!/usr/bin/perl
# Programul CGI principal

require "core.pl";

# preluam preferintele (daca exista)
%prefer = CGI::cookie($cookie_name);

$styles = $prefer{'styles'} || $styles;

print CGI::header();

# antetul paginii
&gen_output($header);

# continutul paginii
print "<h2 class=\"header\">$default_heading</h2>";
&gen_blogs;

# subsolul paginii
&gen_output($footer);
```

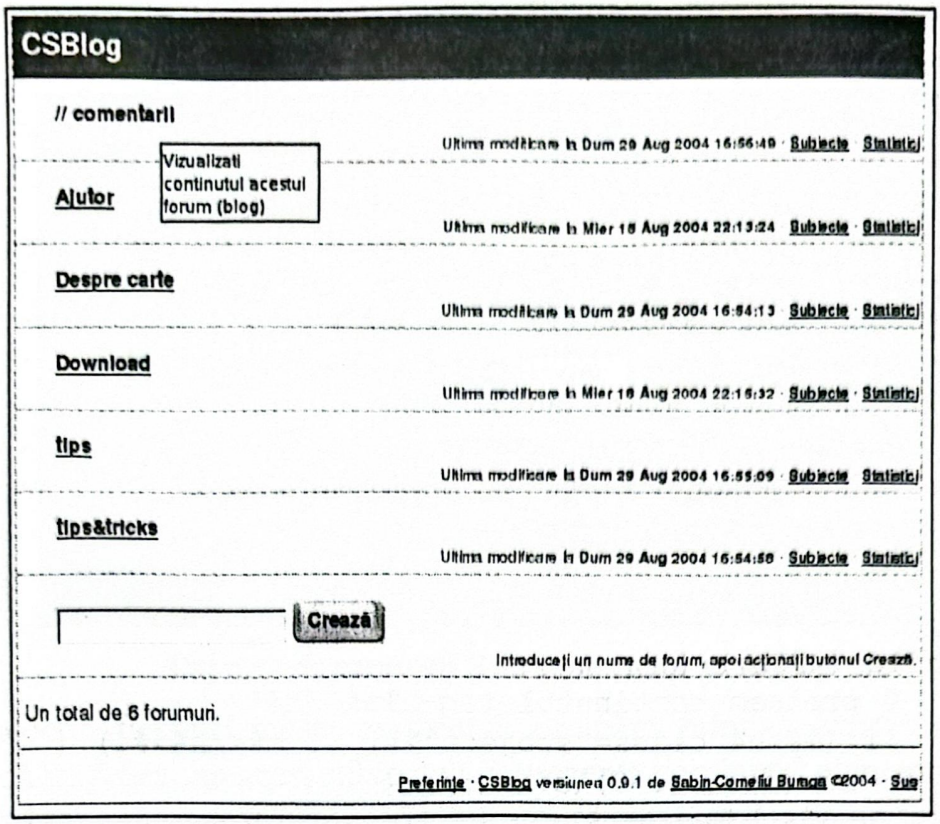
Variabila `$default_heading` indică titlul principal, valoarea sa putând fi modificată prin editarea fișierului *config.pl*. Funcția care generează lista *Weblog*-urilor este definită în *core.pl* și cuprinde următoarele linii Perl:

```
# genereaza lista blog-urilor curente
sub gen_blogs() {
  my @blogs = glob("$blogs/*");
  my $blgs = 0; # numara blog-urile existente

  # listam toate blog-urile (in ordine alfabetica)
  foreach $blog (sort {uc($a) cmp uc($b)} @blogs) {
    # ignoram fisierele ascunse sau
    # pe cele care nu sint directoare
    next if ($blog =~ m/^\./ || not(-d $blog));

    # preluam numele blog-ului
    if ($blog =~ m/$blogs\/(.+)/) {
      $blog_name = $1;
    }
```

```
    print "<div class=\"list\">
      <a href=\"csblog.cgi?topic=" .
      $blog_name . "\"
      title=\"Vizualizati continutul acestui blog\">" .
      CGI::unescape($blog_name) . "</a></div>";
    # afiseaza timpul ultimei modificari
    my @blog_stat = stat($blog);
    print "<div class=\"time\">Ultima modificare la <b>" .
      &get_time(localtime($blog_stat[9])) .
      "</b> &middot; ";
    print "<a href=\"enblog.cgi?topic=" . $blog_name . "\"
      title=\"Vizualizati subiectele mesajelor existente\">
      Subiecte</a> &middot; ";
    print "<a href=\"csblog.cgi?topic=" . $blog_name .
      "&amp;details=yes\"
      title=\"Diverse statistici\">Statistici</a></div>";
    $blgs++;
  }

  # inseram formularul de creare a unui blog
  &gen_file_content($nbform) if $create_blogs;

  print "<div> Un total de " . $blgs .
    (($blgs == 1) ? " forum" : " forumuri") . ".</div>";
}
```

Cum fiecare *blog* este stocat într-un director separat, e uşor să listăm numele directorului şi să generăm o legătură spre scriptul care afişează efectiv conţinutul *blog*-ului în cauză. Pentru aceasta, adăugăm la numele scriptului (*enblog.cgi* sau *csblog.cgi*) şi parametrul `topic`, construind astfel şirul de interogare (*query string*) care va fi transmis – via metoda GET – programului CGI. Folosim `unescape()`, furnizată de modulul CGI, pentru a afişa corespunzător numele codificat al directorului. Pentru a afişa data şi ora ultimei modificări a *blog*-ului (de fapt, a directorului), recurgem la funcţiile standard `stat()` şi `localtime()`, dar şi la funcţia `get_time()`, specificată în *utils.pl*. Această ultimă funcţie generează data în limba română. Funcţia `gen_file_content()` pur şi simplu preia conţinutul unui fişier (în cazul nostru, formularul de creare a unui *blog*) şi-l afişează la ieşirea standard.

Un posibil rezultat al scriptului *index.cgi* poate fi urmărit în figura următoare:



Afişarea listei *Weblog*-urilor curente

3.5 Afişarea subiectelor unui anumit *blog*

Scriptul *enscript.cgi* este răspunzător de crearea listei subiectelor unui anumit *blog*, utilizatorul putându-le sorta cronologic (de exemplu, cel mai vechi va fi primul afişat) sau invers (cel mai recent este primul din listă). De asemenea, se vor genera legături spre conţinutul propriu-zis al mesajului, prin invocarea scriptului *csblog.cgi*.

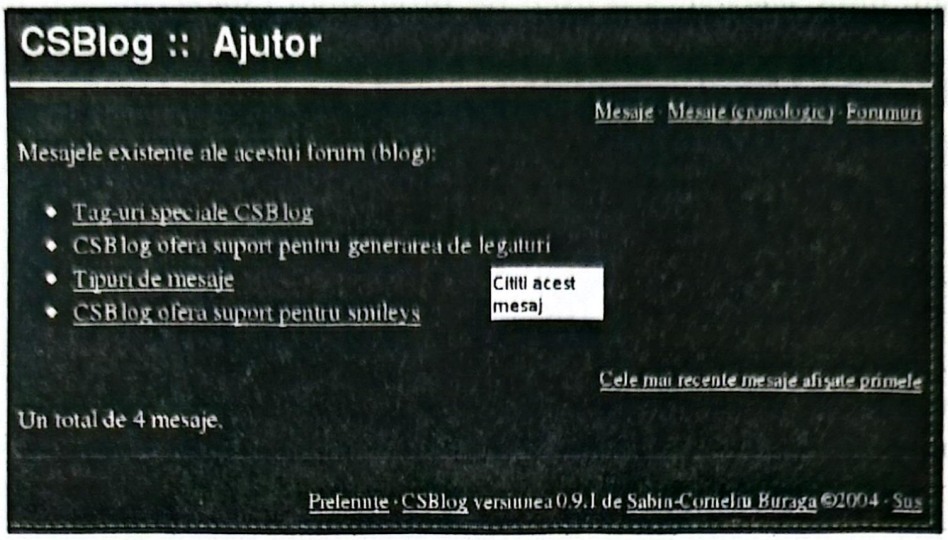
Programul este similar celui anterior, funcţia care implementează operaţiile dorite fiind `gen_blog_entries($topic, $chrono_order)`, unde `$topic` reprezintă numele *blog*-ului, iar `$chrono_order` indică maniera de ordonare a subiectelor. Aceşti doi parametri sunt preluaţi din şirul de interogare al scriptului. Detaliem, în continuare, liniile de cod reprezentative ale funcţiei:

```
# genereaza lista intrarilor unui blog
sub gen_blog_entries() {
  my ($topic, $chrono_order) = @_;

  my $dirname = CGI::escape($topic);

  @messages = glob("$blogs/$dirname/*$suffix");
  @messages = sort {$b cmp $a} @messages
    unless $chrono_order;
```

```
  # nu am gasit nici un mesaj
  unless (scalar(@messages)) {
    print "<div> Acest forum este vid.</div>";
    return;
  }

  print "<div> Mesajele existente:</div>\n";

  # generam lista intrarilor
  print "<ul>";
  foreach $message (@messages) {
    # ignoram fisierele ascunse
    next if $message =~ m/^\./;

    # citim subiectul din fisierul de mesaje
    next unless open(FIL, "<$message");

    while (<FIL>) {
       # preluam continutul tag-ului 'h4'
       if (m#<h4 class="subject">#i .. m#</h4>#i) {
         if (/\:\:(.+?)\s+\:\:/) {
           $subject = $1;
           print "<li><a href=\"csblog.cgi?topic=" .
             $topic . "&amp;chrono=$chrono_order#fragm" .
             $msgs . "\" title=\"Cititi acest mesaj\">
             $subject</a></li>\n";
         }
       }
    }
    close (FIL);
  }
  print "</ul>";

  # oferim posibilitatea schimbarii ordinii de afisare
  # a subiectelor
  my $new_order = 1 - $chrono_order;
  print "<div class=\"time\"><a
    href=\"enblog.cgi?topic=$topic&amp;chrono=$new_order\">";
  if ($chrono_order) {
    print "Cele mai recente";
  }
  else {
    print "Cele mai vechi";
  }
  print " mesaje afi&#351;ate primele</a></div>";
}
```

Trebuie să precizăm aici faptul că, pentru sortare, ne bazăm pe faptul că numele mesajelor reprezintă data și ora la care a fost creat mesajul, așa cum

sunt returnate de `localtime()`, şi codificate cu funcţia `escape()`, pusă la dispoziţie de modulul CGI.



Captură de ecran cu lista subiectelor unui *blog*

Subiectul fiecărui mesaj existent îl preluăm printr-o expresie regulată, ştiind că întotdeauna acest subiect va reprezenta conţinutul elementului `<h4>`. Oferim în continuare structura XHTML a unui mesaj stocat de un fişier *.csblog*:

```
<h4 class="subject">:: Tipuri de mesaje :: scris de
  <a title="Vizitati sau trimiteti un e-mail"
     class="link" href="mailto:busaco@infoiasi.ro">
     Sabin Buraga</a>
</h4>
<div class="message">
<p>Un mesaj poate fi marcat ca fiind
<i>important</i> si/sau <i>interesant</i>.</p>
</div>
<div class="time">Mier 18 Aug 2004 21:05:12</div>
```

Pentru a obţine numai subiectul (adică şirul de caractere delimitat de "::" şi aflat între marcatorii `<h4>` şi `</h4>`), recurgem la liniile de mai jos:

```
if (m#<h4 class="subject">#i .. m#</h4>#i) {
  if (/\:\:(.+?)\s+\:\:/) {
     $subject = $1;
     # variabila $subject contine sirul dorit
     ...
  }
}
```

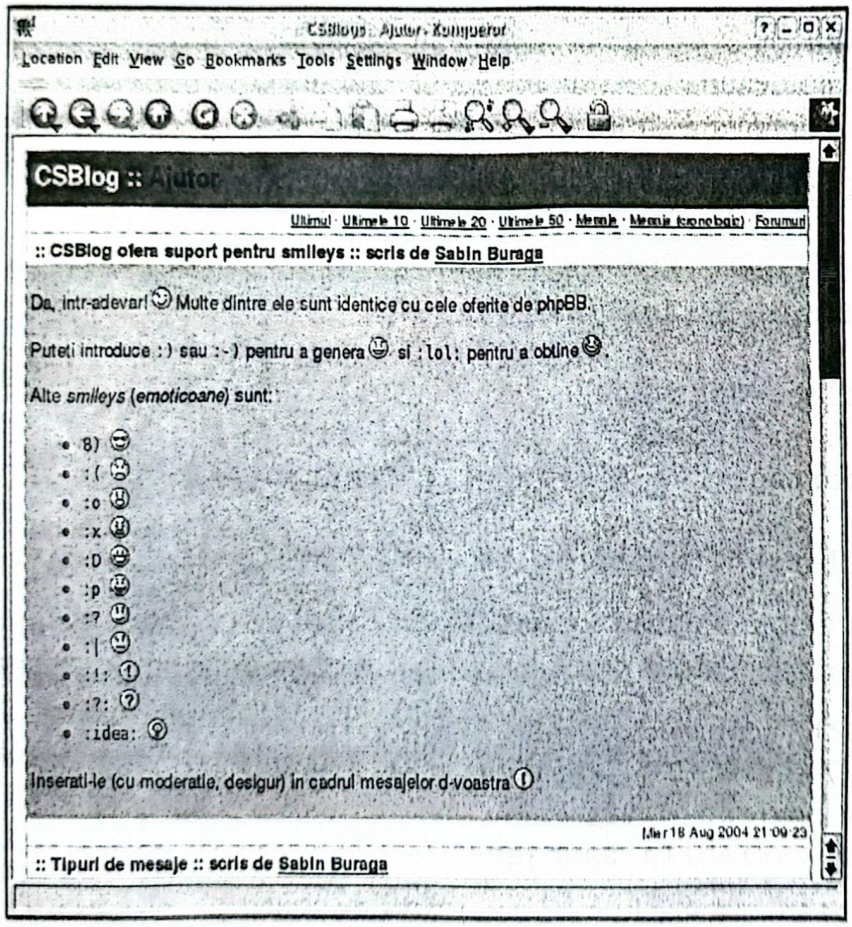
Acest fragment de program ilustrează o utilizare mai puţin întâlnită a operatorului de enumerare „..” în conjuncţie cu expresiile regulate Perl; pentru alte amănunte, consultaţi volumul S. Buraga *et al.*, *Programare Web în bash şi Perl*, apărut în anul 2002 la Editura Polirom.

3.6 Listarea mesajelor unui *blog*

Conținutul unui *Weblog* va fi listat de programul *csblog.cgi*. Mai mult decât atât, acest script va da utilizatorului posibilitatea să introducă un mesaj nou, în cazul în care *blog*-ul este deschis.

Am văzut mai sus că fiecare mesaj generat va fi stocat de un fișier XHTML. Astfel, tot ce trebuie făcut este să preluăm conținutul fiecărui fișier de mesaje și să îl includem în documentul rezultat. Funcția care implementează această operație este numită `gen_content()`. Tot ea va da posibilitatea afișării unui număr mai redus de mesaje (de exemplu, doar cele mai recente 10 sau ultimul redactat). Deoarece codul sursă este relativ simplu, nu-l furnizăm aici.

Discutam mai sus despre posibilitatea afișării unui formular de editare a unui nou mesaj, în cazul în care *blog*-ul nu este închis (blocat). În versiunea discutată aici, acest fapt se realizează verificând existența fișierului *.access* în directorul corespunzător unui *blog* (dacă acest fișier există, atunci considerăm că *blog*-ul este închis).



Parcurgerea conținutului unui *blog*

Funcția care prelucrează câmpurile formularului de introducere a mesajului și care generează fragmentul de cod XHTML corespunzător este numită `add_content()` – vezi *infra*. Formularul va da vizitatorilor posibilitatea să furnizeze subiectul și conținutul mesajului dorit a fi inclus în *blog*, precizând, în plus, identitatea și afilierea acestora. De asemenea, un mesaj

poate fi marcat ca fiind important sau interesant, în aceste cazuri realizându-se, via proprietățile CSS, un alt mod de afișare.

În corpul mesajului, dăm posibilitatea utilizării unor marcatori speciali, delimitați de „[" și „]". Marcatorii (X)HTML nu pot fi folosiți, pentru a restrânge libertatea utilizatorilor de a modifica voit *design*-ul sitului. *Tag*-urile speciale sunt: `[p]` pentru `<p>` (paragraf), `[i]` pentru `<i>` (italic), `[b]` pentru `<b>` (bold), `[pre]` pentru `<pre>` (text preformatat), `[img]` (imagini), `[a]` (ancore – hiperlegături) etc. Acești marcatori speciali sunt similari celor oferiți de forumul *phpBB*. Din punctul de vedere al implementării, recurgem la operatorul de substituție folosit în contextul expresiilor regulate.

De exemplu, legăturile hipertext pot fi incluse via construcția `[a Text]`*URI*`[/a]`. Astfel, pentru a genera:

```
Vizitati-mi <a
href="http://www.infoiasi.ro/~busaco">situl</a>
```

vom putea scrie:

```
Vizitati-mi [a situl]http://www.infoiasi.ro/~busaco[/a]
```

După „[a", va trebui adăugat, obligatoriu, măcar un spațiu, pentru a obține o legătură.

De asemenea, vom oferi facilități de generare a emoticonurilor (pictograme sugerând stări emoționale, precum veselia sau tristețea). Aceste pictograme vor fi inserate în locul unor succesiuni de simboluri speciale. De exemplu, „`:)`" sau „`:-)`" va genera un fișier grafic reprezentând un chip vesel.

Mai mult decât atât, scriptul va detecta apariția unei adrese Web corecte și automat va insera un *hyperlink* spre aceasta. De exemplu, dacă utilizatorul va introduce:

```
Vizitati www.infoiasi.ro
```

în cadrul mesajului final se va obține:

```
Vizitati <a href="http://www.infoiasi.ro"
title="Deschide acest URI intr-o fereastra noua"
target="_blank">www.infoiasi.ro</a>
```

Același mecanism va fi aplicat și pentru adresele prefixate de schema `http://`.

Procesarea „inteligentă" a URI-urilor este disponibilă și pentru afilierea utilizatorului (acesta are atașată o legătură pentru adresa sa de *e-mail* sau cea Web). Vom putea scrie doar `busaco@infoiasi.ro` și schema de tip *e-mail* (`mailto:`) va fi în mod automat generată.

Listing-ul funcției `add_content()` – inclusă în fișierul *core.pl* – este următorul, unde parametrii de intrare reprezintă:

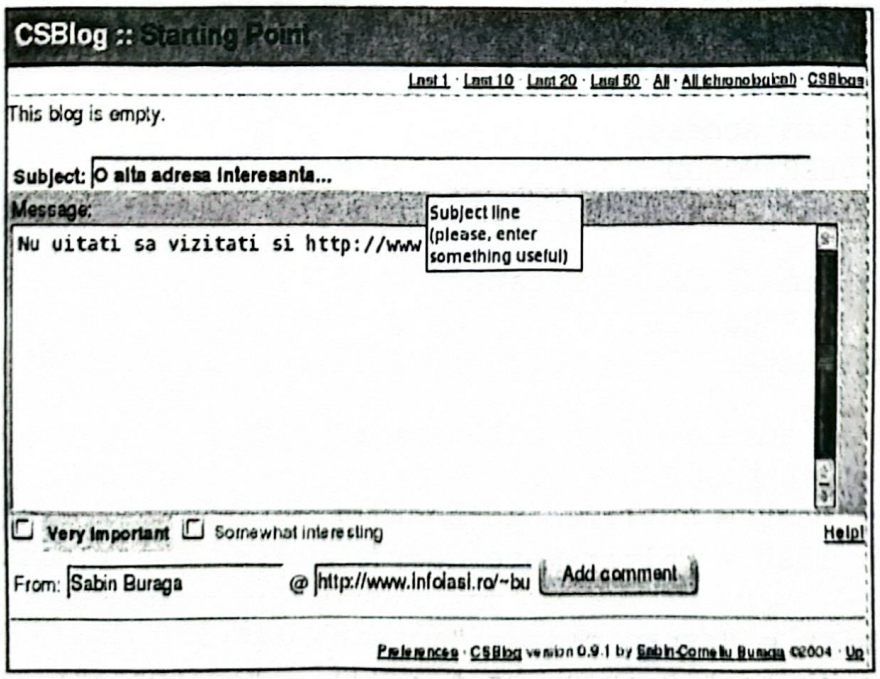
- `$topic` – numele *blog*-ului unde va fi inserat mesajul;
- `$edit` – corpul (obligatoriu) al mesajului;
- `$subject` – subiectului mesajului;

- `$name` – numele utilizatorului care a redactat mesajul;
- `$uri` – afilierea utilizatorului (adresă de *e-mail* sau Web);
- `$important` – mesajul va fi considerat important;
- `$interesting` – mesajul va fi tratat ca fiind interesant.

```
# adauga continutul unui mesaj intr-un fisier XHTML
sub add_content() {
 my ($topic, $edit, $subject, $name,
     $uri, $important, $interesting) = @_;

 # procesarea 'inteligenta' a corpului mesajului
 $_ = CGI::escapeHTML($edit);

 # inseram 'smileys'
 s{(\s*\:\-?\)\s*)}
   {<img src="pics/icon_smile.gif"
         alt="Zambitor :-)" />}gs;
 s{(\s*\;\-?\)\s*)}
   {<img src="pics/icon_wink.gif"
         alt="Smecher ;-)" />}gs;
 s{(\s*\:\-?o\s*)}
   {<img src="pics/icon_surprised.gif"
         alt="Surprins :-o" />}gsi;
 s{(\s*\:lol\:\s*)}
   {<img src="pics/icon_lol.gif"
         alt="Ras copios" />}gs;
 # ...si altele similare

 # includem tag-urile speciale '[]'
 s{\[pre\](.+?)\[\/pre\]} {<pre>$1</pre>}gs;
 s{\[b\](.+?)\[\/b\]} {<b>$1</b>}gs;
 # ...si altele similare
 s{\[img\](.+?)\[\/img\]}
   {<img src="$1" alt="Imagine: $1" />}gs;

 # inseram hyperlink daca apare '[a]..[/a]'
 unless (s{\[a\s+(.+?)\](.+?)\[\/a\]}
   {<a href="$2"
      title="Deschide acest URI intr-o fereastra noua"
      target="_blank">$1</a>}gs) {
   # inseram hyperlink daca detectam 'www.' ori 'http://'
   s{(http\:\/\/)(\S+)((\s+)|$)}
     {<a href="http://$2"
       title="Deschide acest URI intr-o fereastra noua"
       target="_blank">$2</a>$3}gs ||
       s{(www\.)(\S+)((\s+)|$)}
       {<a href="http://www\.$2"
         title="Deschide acest URI intr-o noua fereastra"
```

```
        target="_blank">www\.$2</a>$3)gs;
}

$edit = $_;

# procesare 'inteligenta' a URI-ului
# verificam daca e o adresa e-mail valida
if (&valid_email($uri)) {
  $uri = "mailto:$uri";
}
else {
  # verificam daca e un URL (incepe cu 'www.')
  if ($uri =~ /^www\.\S+/) {
    $uri = "http://$uri";
  } else {
    $uri = CGI::escapeHTML($uri);
  }
}

my $basename = CGI::escape(scalar(localtime));
my $dirname = CGI::escape($topic);

# cream fisierul
unless (-d "$blogs/$dirname" && -r "$blogs/$dirname") {
  # a aparut o eroare la creare
  return $ERR_NOCREATE
    unless $create_blogs;
  # daca e primul mesaj din blog,
  # cream si directorul acestuia
  mkdir "$blogs/$dirname" ||
    return $ERR_NOCREATE;
}

# verificam accesul
return $ERR_NOADD
  if &check_access("$blogs/$dirname/$access");

# deschidem fisierul creat
return $ERR_NOOPEN
  unless (open (FIL, ">$blogs/$dirname/$basename$suffix"));

# retinem adresa/gazda utilizatorului
if ($retain_host) {
  $user_host = CGI::remote_host();
  $user_host = CGI::remote_addr() unless $user_host;
}
$user_host = "<span class=\"author\">(" .
  ($retain_host ? $user_host : '') . ")</span>";
```

```
# generam codul XHTML...
$edit = "<div class=\"interesting\">$edit</div>"
  if ($interesting);

if ($important) {
  print FIL "<h4 class=\"subject\">::
    <span class=\"important-subject\">" .
    CGI::escapeHTML($subject) .
    "</span> :: scris de <a  class=\"link\" href=\"" .
    $uri . "\">" . CGI::escapeHTML($name) .
    "</a> $user_host</h4>\n";

  print FIL "<div class=\"important\">";
}
else {
  print FIL "<h4 class=\"subject\">:: " .
    CGI::escapeHTML($subject) .
    " :: scris de <a class=\"link\" href=\"" .
    $uri . "\">" . CGI::escapeHTML($name) .
    "</a> $user_host</h4>\n";

  print FIL "<div class=\"message\">";
}
print FIL "$edit</div>\n";
print FIL "<div class=\"time\">" .
  &get_time(localtime) . "</div>\n";
close (FIL);

return 0;
}
```



Formularul de editare a unui mesaj

Funcţia de verificare a validităţii unei adrese de poştă electronică – denumită `valid_email()` – recurge tot la expresii regulate, fiind stocată în fişierul *utils.pl*. Implementarea ei o lăsăm ca exerciţiu pentru cititor.

La finalul acestei secţiuni, vom furniza codul scriptului *csblog.cgi*:

```
#!/usr/bin/perl
# Afiseaza continutul unui blog

require "core.pl";

# preluam preferintele din cookie
%prefer = CGI::cookie($cookie_name);

$styles = $prefer{'styles'} || $styles;

# preluam campurile formularului:
# corpul mesajului
$text = CGI::param('text');
# subiectul mesajului
$subject = CGI::param('subject')
   || $default_subject;
# numele vizitatorului
$name = CGI::param('name');
# adresa (afilierea) vizitatorului
$uri = CGI::param('uri');
# mesaj important
$important = CGI::param('important');
# mesaj interesant
$interesting = CGI::param('interesting');
# numarul de intrari afisate
$entries = CGI::param('entries');
# numele blog-ului
$topic = CGI::param('topic')
   || $default_topic;
# ordinea de afisare a mesajelor
$chrono_order = CGI::param('chrono');

# vedem daca exista deja stocate in cookie
# numele si afilierea utilizatorului
unless ($name) {
   $name = $prefer{'name'} || $default_name;
}
unless ($uri) {
   $uri = $prefer{'uri'} || $default_uri;
}

$method = CGI::request_method();
```

```
# verificam daca datele au fost obtinute
# prin metoda POST; doar in acest caz, adaugam mesajul
if ($method =~ m/post/i && $text) {
  $status = &add_content ($topic, $text, $subject,
    $name, $uri, $important, $interesting);
  # raportam erorile
  if ($status == $ERR_NOCREATE) {
    &print_error("Crearea unui blog e interzis&#259;.");
    exit;
  } else {
    if ($status == $ERR_NOOPEN) {
      &print_error("Regret&#259;m,
         a intervenit o eroare.");
      exit;
    }
    else {
      if ($status == $ERR_NOADD) {
        &print_error("Regret&#259;m,
           dar acest forum e &icirc;nchis.
           Nu pute&#355;i insera mesaje.");
        exit;
      }
    }
  }
  # redirectam, sa afisam mesajele blog-ului
  print "Location: csblog.cgi?entries=$entries" .
    "&topic=$topic\n\n";
  exit;
}

print CGI::header();

# antetul paginii
&gen_output($header);

# generam continutul
print "<h2 class=\"header\">$default_heading ::
<span class=\"subject\">$topic</span></h2>";

&gen_blog_content($topic, $entries, $chrono_order);

# subsolul paginii
&gen_output($footer);
```

Funcția `print_error()` are rolul de a afișa utilizatorului posibilele erori survenite. Codul ei, fiind foarte simplu, nu va mai fi prezentat aici.

4. Alternative și posibile extinderi viitoare

O primă alternativă la implementarea prezentată în cadrul subcapitolului anterior o constituie utilizarea unui format mai flexibil de stocare a mesajelor. Desigur, ne gândim la limbajul XML. Folosind modulele `XML::Writer` sau `XML::Simple`, avem posibilitatea generării comode de elemente și atribute XML, iar cu `XML::Parser` ori `XML::SAX` putem procesa mesajele pentru a genera codul XHTML corespunzător.

O altă soluție ar fi să utilizăm modelul DOM (vezi modulul `XML::DOM`) sau să realizăm transformări XSLT, astfel încât să putem genera ieșiri alternative, în funcție de navigatorul folosit. Pentru această ultimă metodă, putem folosi modulul `XML::XSLT` sau `XML::Sablotron`.

Aplicația ar putea fi extinsă, adăugând o componentă de generare a unui fișier RSS conținând subiectele ultimelor mesaje adăugate. De asemenea, se pot prelua date RSS oferite de alte situri, pentru aceasta având la dispoziție modulele `XML::RSS` și `LWP::Simple`.

În vederea reținerii permanente a informațiilor privind vizitatorii care au scris mesaje pe diverse *blog*-uri și, eventual, a transformării aplicației *CSBlog* într-un gen de forum, putem recurge la stocarea datelor fie într-o bază de date convențională (MySQL ori PostgreSQL), fie într-una nativă XML. În primul caz, putem folosi interfața pusă la dispoziție de modulele `DBI`, iar în al doilea, recurgem la transfer de date XML prin SOAP, utilizând `SOAP::Lite`.

Încurajăm cititorii interesați să experimenteze diverse (alte) alternative și extinderi ale implementării curente sau să încerce integrarea acestei aplicații în cadrul unui portal sau al unui instrument de colaborare disponibil pe Web (de exemplu, *TWiki – http://TWiki.org*).

5. Comentarii bibliografice

Informațiile referitoare la fenomenul *Weblogging*-ului mondial au fost preluate, în principal, din volumul C. Doctorow *et al.*, *Essential Blogging*, O'Reilly & Associates, 2002 și din „The future of Weblogging" de N. Macdonald, articol disponibil *on-line* la *http://www.theregister.co.uk/2004/04/18/blogging_future/*. Un alt articol foarte interesant este cel semnat de Mircea Sârbu și intitulat „Comunități: pluralul singularului" – a se consulta adresa *http://www.intraweb.ro/txt/Articole/Weboscop/Blogs/show*.

Pentru inițierea și aprofundarea programării în Perl (mai ales în contextul Web-ului), recomandăm consultarea cărților S. Buraga, V. Tarhon-Onu, Ș. Tanasă, *Programare Web în bash și Perl*, Editura Polirom, Iași, 2002 și Ș. Trăușan-Matu *et al.*, *Prelucrarea documentelor folosind XML și Perl*, Editura Matrix Rom, București, 2001. Două situri de real interes sunt *http://*

www.perl.com şi *http://www.perlmonks.com.* Mai pot fi vizitate şi *http://www.onlamp.com* sau *http://www.perlxml.com.*

Diverse module Perl de procesare a documentelor XML pot fi căutate şi instalate urmând adresa *http://www.cpan.org/modules/by-module/XML.*

În ceea ce priveşte formatul RSS, recomandăm tutorialul disponibil *on-line* la *http://www.mnot.net/rss/tutorial/* şi studierea documentelor RSS generate de diverse situri, precum *http://slashdot.org/slashdot.rdf* sau *http://www.xml.com/xml/news.rdf.* Utilizarea tehnologiei RSS în contextul Web-ului semantic poate fi urmărită în cartea *Semantic Web. Fundamente şi aplicaţii,* apărută în 2004 la Editura Matrix Rom şi avându-l drept autor pe Sabin Buraga (detalii la adresa *http://www.infoiasi.ro/~sweb/*).

CAPITOLUL VIII

Hattrick – simularea meciurilor de fotbal pe Web

Daniel Puiu, daniel.puiu@gmail.com
Marian Bostan, marianinterist81@yahoo.com

În cadrul acestui capitol este prezentată o modalitate de simulare a unui meci de fotbal dintre două echipe virtuale. Pe lângă descrierea detaliată a algoritmilor folosiți, se insistă și asupra tehnologiilor folosite, expunându-se interoperabilitatea limbajului C# cu Microsoft SQL Server și XML. Simularea meciului este integrată apoi într-un manager de fotbal disponibil pe Web, oferindu-se și un client *desktop* bazat pe servicii Web.

1. Fotbalul, între realitate și simulare

Ce sport poate fi mai plăcut decât fotbalul? Indiferent dacă este practicat la nivel profesionist sau amator, urmărit din tribune sau în fața televizorului, fotbalul a fost, este și va fi unul dintre cele mai iubite sporturi din lume.

Popularitatea de care se bucură în prezent s-a manifestat și în alte domenii, unul dintre acestea fiind și industria software. Companiile de jocuri pe calculator au exploatat această piață, apărând astfel numeroase jocuri bazate pe această temă. Seria *Fifa 98, 2000,... 2004* s-a bucurat de un real succes în întreaga lume.

Trebuie făcută însă o diferență clară între acestea și jocurile de tipul *Championship Manager*. În primul caz, utilizatorul își conduce echipa în timpul meciurilor, accentul punându-se pe îndemânarea sa în deplasarea jucătorilor pe teren. Calitățile acestora nu influențează într-un mod determinant rezultatul meciului, un adevărat cunoscător putând obține rezultate remarcabile și cu o echipă mai slab cotată. Bineînțeles, un rol decisiv îl deține și gradul de inteligență artificială – *AI* (*Artificial Intelligence*) – a jocului, care poate să mărească sau, dimpotrivă, să scadă nivelul de realitate din cadrul jocului. Modulul grafic este dezvoltat corespunzător cerințelor unui astfel de joc.

Pentru jocurile de tip *manager*, se insistă pe atribuţiile de manager al unui club de fotbal, cu toate îndatoririle care decurg de aici. Managementul financiar, realizarea transferurilor, coordonarea personalului tehnic, întreţinerea stadionului sau programarea meciurilor amicale sunt doar câteva dintre acestea. Dar adevărata atracţie o reprezintă sarcina de antrenor al echipei. Cu toate că, în realitate, managerul nu este şi antrenor, această funcţie reprezintă adevărata provocare pentru toţi utilizatorii. Studierea adversarului, selectarea jucătorilor care vor intra în componenţa echipei, stabilirea poziţiilor din teren, repartizarea atribuţiilor etc. sunt sarcini pe care orice antrenor trebuie să le îndeplinească pentru a asigura succesul echipei sale. În acest caz, partea grafică şi, implicit, prezentarea meciului diferă de prima categorie, accentul fiind pus pe prezentarea informaţiilor utile din meci, pentru ca antrenorul să ia cele mai bune decizii.

Indiferent de tipul jocului, aplicaţiile de acest gen vor continua să fascineze şi de acum înainte, tehnologia actuală permiţând o dezvoltare continuă şi, implicit, creşterea gradului de realitate din cadrul simulării.

2. Simularea meciului – prezentare generală

Subcapitolul de faţă descrie o modalitatea de a simula un meci de fotbal pornindu-se de la două loturi de jucători avute la dispoziţie, urmând ca, în subcapitolul următor, să se prezinte o implementare efectivă a unui manager de fotbal pe Web, care va avea ca nucleu acest modul.

Informaţiile privitoare la cele două echipe sunt memorate în şase tabele principale: *teams*, *players*, *matches*, *skills*, *orders* şi *training*. Pentru a putea folosi cu uşurinţă câmpurile şi pentru evitarea confuziilor, numele acestora au fost prefixate de prima literă din numele tabelei. Structura tabelelor se găseşte ca anexă la finalul acestui capitol.

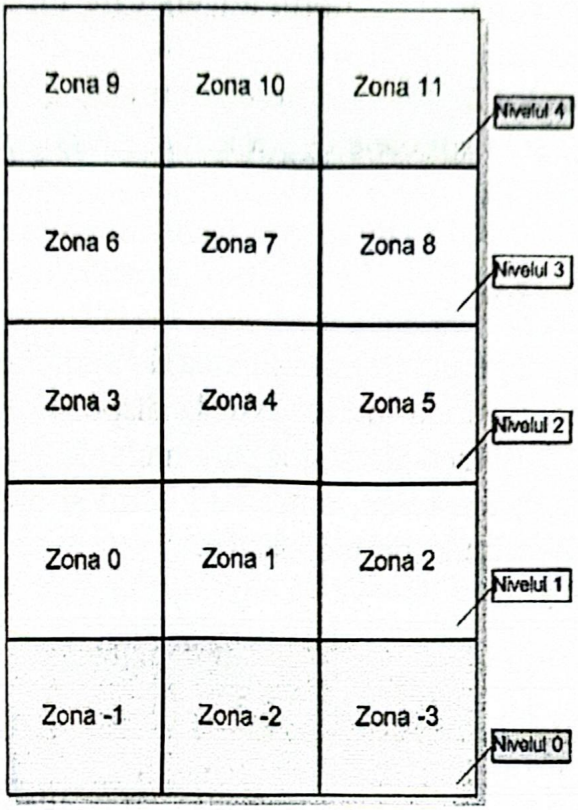
2.1 Zonele terenului

Cum altfel am putea să începem decât cu descrierea terenului de fotbal virtual? Similar fotbalului real, terenul este împărţit pe zone, având însă o semnificaţie diferită. Aceste zone pot fi considerate drept interfaţă între jucători (reprezentaţi de calităţile acestora) şi meci (simularea jocului).

Terenul este divizat în 15 zone numerotate de la –3 la 11, dintre care doar 9 zone sunt considerate zone de joc efectiv (0-8). Acestea au fost grupate în 5 niveluri, câte 3 zone pe fiecare nivel. Să vedem, în continuare, semnificaţiile acestora şi rolul lor în simularea meciului:

- Zonele 0-2 formează nivelul 1, pe care sunt repartizaţi fundaşii echipei gazde şi atacanţii echipei oaspete;

- Zonele 3-5 alcătuiesc nivelul 2 și reprezintă mijlocul terenului – aici sunt repartizați mijlocașii ambelor echipe (zone neutre);
- Evident, zonele 6-8 formează cel de-al treilea nivel, care poate fi considerat opus nivelului 1, aici fiind distribuiți atacanții echipei gazde și fundașii echipei vizitatoare.



Împărțirea terenului pe zone și niveluri

Spre deosebire de fotbalul real, unde jocul poate fi reluat și din apropierea propriei porți, în cazul nostru, meciul se va relua întotdeauna dintr-o zonă centrală (de pe nivelul 2). O situație particulară o reprezintă startul unei reprize sau înscrierea unui gol, caz în care meciul se reia din zona 4. Desfășurarea meciului poate fi văzută ca o succesiune de zone, care începe cu una dintre zonele nivelului 2 (partea centrală a terenului) și continuă cu celalalte zone, până la apariția unui eveniment (pe care îl vom descrie ulterior). În aceste zone au loc „luptele" pentru balon, echipa învingătoare avansând într-o zonă superioară, cealaltă retrăgându-se în apărare. Implementarea algoritmului care stabilește echipa câștigătoare pentru zona respectivă este unul dintre principalele noastre scopuri.

Să considerăm următorul scenariu: meciul începe din zona 4. Aici are loc o confruntare între mijlocașii celor două echipe; să presupunem că gazdele înving. În acest moment, acestea avansează pe următorul nivel (de exemplu, în zona 6), astfel jocul mutându-se pe partea dreaptă a apărării oaspeților. Dacă și de această dată gazdele reușesc să învingă, trecând de apărarea adversă, se ajunge la ultimul nivel (nivel *critic*), aici având confruntarea

finală între portarul oaspeţilor şi un jucător de-al gazdelor. În caz contrar, se revine la o zonă anterioară şi jocul continuă.

Nivelul critic, amintit mai sus, este reprezentat de unul din nivelurile 0 şi 4, fiind format din zonele –3, –2, –1 sau 9, 10, 11. Acestea nu sunt zone de teren în adevăratul sens al cuvântului, ele fiind folosite doar teoretic în stabilirea învingătorului dintre portar şi jucătorul advers care încearcă să înscrie.

După ce ne-am format o idee despre modul de desfăşurare a meciului, prin prezentarea zonelor de teren, vom continua cu prezentarea formaţiilor de joc, urmând ca, în subcapitolele ulterioare, să descriem detaliat algoritmul de simulare.

2.2 Formaţiile de joc

Ca orice antrenor, managerul are posibilitatea de a trimite în teren o formulă de echipă care să corespundă lotului avut la dispoziţie, dar să răspundă şi tacticii adversarului. Deşi tacticile de joc sunt mult mai numeroase în fotbalul real, pe lângă aşezarea din teren, antrenorii dând şi ordine individuale, în cazul nostru avem 7 formule de joc clasice.

Denumire	Portari	Fundaşi	Mijlocaşi	Atacanţi
3-4-3	1	3	4	3
3-5-2	1	3	5	2
4-5-1	1	4	5	1
4-4-2	1	4	4	2
4-3-3	1	4	3	3
5-3-2	1	5	3	2
5-4-1	1	5	4	1

Formulele de joc

Pentru fiecare dintre cele trei niveluri principale, avem stabilit numărul de jucători. Ele pot fi identificate cu apărarea, mijlocul şi atacul unei echipe de fotbal real. Deşi, în realitate, formulele de joc sunt mult mai complexe, în cazul nostru vom considera că pe fiecare nivel se va găsi un număr fix de jucători.

Cei 11 jucători sunt distribuiţi în teren astfel:

- pe nivelul 1 (zonele 0, 1, 2) sunt poziţionaţi fundaşii;
- pe nivelul 2 (zonele 3, 4, 5) sunt poziţionaţi mijlocaşii;
- pe nivelul 3 (zonele 6, 7, 8) sunt poziţionaţi atacanţii.

Bineînţeles, echipa are întotdeauna un singur portar, acesta neinfluenţând nici o zonă a terenului. El va avea un rol special în cadrul meciului şi, de aceea, despre el vom vorbi separat.

Ordinea de apariție a jucătorilor în listă (structura de date care memorează identificatorii jucătorilor) este foarte importantă, deoarece ea determină pe ce posturi vor fi distribuiți jucătorii. Dacă formula de joc este *a-b-c*, atunci vom avea:

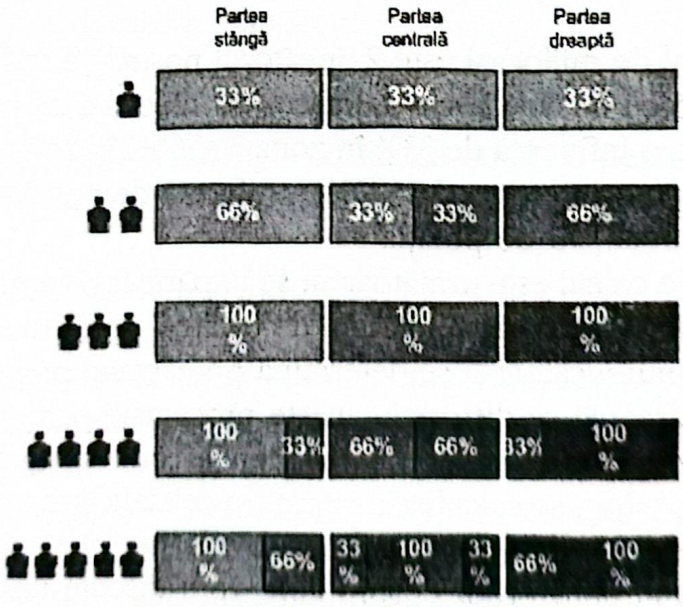
- portar – primul jucător;
- fundași – următorii *a* jucători;
- mijlocași – următorii *b* jucători;
- atacanți – ultimii *c* jucători.

Și totuși, cum influențează formulele de joc desfășurarea meciului? Fiecare jucător va fi repartizat într-o anumită *parte* a terenului. Am spus parte, și nu zonă, pentru că aceasta poate acoperi mai mult de o zonă. În funcție de numărul acestora este determinată contribuția fiecăruia pe fiecare zonă. Un jucător nu poate avea influențe în mai mult de două zone; el întotdeauna va influența cel puțin o zonă și cel mult două zone vecine. Există însă o excepție: dacă nivelul respectiv nu are decât un singur jucător, acesta va influența fiecare zonă cu 33% (o treime din influența totală de pe acel nivel).

Inițial, fiecare jucător are o contribuție de 0% în fiecare zonă a terenului. Apoi, ținând cont de numărul jucătorilor care se află pe acel nivel (număr care poate varia de la 1 la 5), se determină cu ce procent din calitățile sale va influența fiecare zonă. Condiția ca un jucător să aibă un procentaj maxim pe o anumită zonă este ca numărul jucătorilor de pe nivel să fie mai mare sau egal cu 3.

Repartiția ar putea fi examinată din trei puncte de vedere:

- numărul de jucători de pe nivel este 3, caz în care fiecare are o contribuție de 100% pe câte o zonă;
- numărul de jucători de pe nivel este mai mic decât 3 – jucătorii influențează și zone vecine;
- numărul de jucători este mare decât 3 – sunt zone influențate de câte doi jucători.



Repartiția jucătorilor pe zone

Să considerăm un exemplu. Presupunem că dorim să calculăm influența pe zone pentru al doilea mijlocaș. Astfel, în cel de-al doilea *switch* sunt testate toate cazurile referitoare la numărul de mijlocași.

```
switch (mid) //pozitia in teren - mijlocas
{
   case 2:    //al doilea mijlocas
      switch (tp.nMid) //numarul total de mijlocasi
      {
         case 2: //avem in total 2 mijlocasi
            //influenteaza zona 4 cu 33% si zona 5 cu 66%
            tp.play[i].infl[4] = 100 / (float)3;
            tp.play[i].infl[5] = 200 / (float)3;
            break;
         case 3:
            tp.play[i].infl[4] = 100;
            break;
         case 4: //in total sunt 4 mijlocasi
            tp.play[i].infl[3] = 100 / (float)3;
            tp.play[i].infl[4] = 200 / (float)3;
            break;
         case 5:
            tp.play[i].infl[3] = 200 / (float)3;
            tp.play[i].infl[4] = 100 / (float)3;
            break;
         default:
            break;
      }
      break;
   ...
}
```

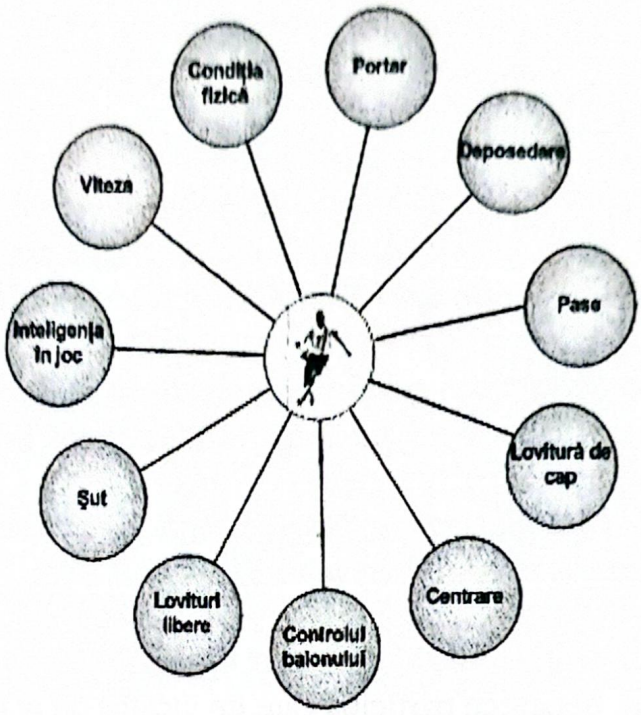
Dacă numărul de mijlocași este 2, jucătorul nostru va influența zona 4 cu 33.33% și zona 5 cu 66.66%. Dacă numărul de mijlocași de pe nivel este însă 4, acesta va avea o influență de 33% în zona 3 și 66% în zona 4. Deci, în total, el are întotdeauna o influență de 100%, indiferent de numărul de jucători de pe respectivul nivel sau de poziția sa.

Modalitatea de calcul este următoarea: se împarte la 3 (numărul de zone de pe nivel) rezultatul înmulțirii numărului de jucători de pe nivelul respectiv cu 100 (punctajul unui jucător) și se determină procentajul pentru o zonă. Apoi, de la stânga la dreapta se distribuie aceste procentaje pentru fiecare zonă în parte. Fiecare zonă va avea același procentaj, însă obținut de la jucători diferiți. În mod analog, se calculează și pentru celelalte scheme de distribuire.

Aceste procentaje vor fi folosite la determinarea punctajelor pe zone în funcție de calitățile jucătorilor despre care vom discuta în continuare.

2.3 Calitățile jucătorilor

Diferența dintre două echipe în timpul meciului este făcută în primul rând de jucătorii din teren. Cu toate că, în fotbalul real, echipele care se bazează pe unitatea grupului și pe relațiile dintre jucători, mai puțin pe individualități, au început să aibă rezultate răsunătoare (vezi victoria Greciei la Campionatul European de Fotbal din 2004), valorile individuale pot face diferența din teren.



Calitățile jucătorilor

În cazul nostru, jucătorii au fost „înzestrați" cu cele mai importante calități care se regăsesc și la jucătorii adevărați:

- *portar* (`keep` – *keeping*) – această calitate va influența doar evoluția portarului;
- *deposedare* (`tack` – *tackling*) – calitatea de bază pentru fundași;
- *pase* (`pass` – *passing*) – se găsește la toate cele trei categorii de jucători de câmp;
- *lovitura de cap* (`head` – *heading*) – similar cu *pase*, dar cu influențe mai mici;
- *centrări* (`cbal` – *cross balls*) – calitate importantă doar pentru mijlocași;
- *controlul balonului* (`bcon` – *ball control*) – influențează în special atacul și mjlocul;
- *lovituri libere* (`spi` – *set pieces*) – singura calitate necesară pentru executarea loviturilor libere și a penaltiurilor;
- *șut* (`sho` – *shoot*) – esențială pentru înscrierea unui gol;

- *inteligența în joc* (`pint` – *player intelligence*) – una dintre cele mai importante calități, prezentă în toate compartimentele;
- *viteza* (`speed` – *speed*) – importantă pentru atac;
- *condiția fizică* (`sta` – *stamina*) – calitate de bază pentru randamentul mijlocașilor.

Transpuse într-o metrică, fiecare calitate poate avea o valoare de la 0 (nu este prezentă la jucător) până la 100 (valoarea maximă). Aceste valori sunt memorate în tabela *skills*.

Pentru a vedea cum sunt folosite calitățile jucătorilor efectiv în simularea meciului, trebuie să vedem mai întâi tipurile de confruntări care vor avea loc între jucători:

- *confruntarea jucători gazde-jucători oaspeți* pentru câștigarea balonului poate apărea în una dintre zonele nivelurilor 1, 2 sau 3. Mai poate fi denumită și *confruntare zonală*, la ea luând parte mai mulți jucători. În zona respectivă poate avea loc una dintre următoarele „subconfruntări":
  - *apărare* (*defence*) *echipa 1-atac* (*attack*) *echipa 2*, corespunzător nivelului 1 dacă *echipa 1* este echipa gazdă sau nivelului 3, în caz contrar;
  - *mijloc* (*middle*) *echipa 1-mijloc* (*middle*) *echipa 2*, pentru partea mediană a terenului (nivelul 3);
- *confruntarea portar-jucător advers* apare în cadrul unei ocazii de gol la unul dintre nivelurile 0, 1, 3 sau 4, fiind denumită și confruntare *individuală*, deoarece participă câte un jucător de la fiecare echipă. La rândul ei, poate fi una dintre „subconfruntările":
  - *ocazie de gol din acțiune* – confruntare între portar (*keeping*) și un jucător advers (*scoring*) pe unul dintre nivelurile 0 sau 4;
  - *ocazie de gol* (*penalty*) – confruntare între portar (*keeping penalty*) și un jucător advers (*scoring penalty*) pe unul dintre nivelurile 0 sau 4;
  - *ocazie de gol* (*lovitură liberă*) – confruntare între portar (*keeping freekick*) și un jucător advers (*scoring freekick*) pe unul dintre nivelurile 1 sau 3.

Dintr-o asemenea confruntare, este evident că numai jucătorii unei singure echipe vor ieși câștigători. În cazul unei confruntări individuale, jucătorii vor influența faza respectivă exclusiv prin calitățile pe care le au. În cealaltă situație, este luat în calcul și procentajul de acoperire a zonei respective. Să remarcăm, în următorul tabel, modul în care influențează calitățile jucătorilor confruntările descrise mai sus; despre algoritmul de determinare a învingătorului vom vorbi mai târziu.

Tipul confruntării	*Keep*	*Tack*	*Pass*	*Head*	*Cbal*	*Bcon*	*Spi*	*Sho*	*Pint*	*Spe*	*Sta*
Attack	0	0	20	15	0	20	0	0	20	20	5
Middle	0	5	10	0	10	25	0	0	25	5	20
Defence	0	30	20	15	0	10	0	0	10	10	5
Scoring penalty	0	0	0	0	0	0	100	0	0	0	0
Scoring freekick	0	0	0	0	0	0	100	0	0	0	0
Scoring	0	0	0	20	0	15	0	50	15	0	0
Keeping penalty	100	0	0	0	0	0	0	0	0	0	0
Keeping freekick	100	0	0	0	0	0	0	0	0	0	0
Keeping	100	0	0	0	0	0	0	0	0	0	0

Influența calităților jucătorilor asupra tipurilor de confruntări

Se poate observa că un mijlocaș trebuie, în primul rând, să aibă un bun control al balonului (`cbal = 25`) și o bună inteligență în joc (`pint = 25`), apoi condiția fizică (`sta = 20`) este foarte importantă, și, în cele din urmă, pasele, centrările, deposedarea și viteza. În mod similar, se determină și procentajele pentru celelalte tipuri de confruntări. Pentru a nu fi defavorizată o echipă sau, mai bine spus, un oponent dintr-o confruntare, suma procentajelor de pe fiecare „subconfruntare" este întotdeauna 100. Deci diferențierea se va face după modul de acoperire a zonelor respective de către jucători și după calitățile acestora.

Trebuie precizat faptul că toate aceste valori reprezintă procentaje din calitatea jucătorului respectiv. Astfel, dacă avem un fundaș cu calitatea `pase` egală cu 60, atunci în apărare el va obține 60 * 20% = 12 puncte cu această calitate (`def = 20`). Modalitatea exactă de calcul al punctajelor obținute de jucători va fi descrisă în secțiunea următoare.

Tabelul de mai sus este memorat într-un fișier XML, valorile putând fi modificate ulterior pentru o mai bună corespondență cu fotbalul real. Înainte

de începerea generării meciului, fişierul este procesat, iar informaţiile obţinute vor fi folosite pentru calcularea punctajelor.

```
<?xml version="1.0" encoding="utf-8" ?>
<players>
      <position type="attack">
            <keep>0</keep>
            <tack>0</tack>
            <pass>20</pass>
            <head>15</head>
            <cbal>10</cbal>
            <bcon>25</bcon>
            <spi>0</spi>
            <sho>0</sho>
            <pint>20</pint>
            <spe>20</spe>
            <sta>5</sta>
      </position>
      <position type="middle">
            <keep>0</keep>
            <tack>5</tack>
            <pass>10</pass>
            <head>0</head>
            <cbal>10</cbal>
            <bcon>15</bcon>
            <spi>10</spi>
            <sho>5</sho>
            <pint>25</pint>
            <spe>5</spe>
            <sta>20</sta>
      </position>
      ...
</players>
```

2.4 Simularea meciului

După ce am descris repartiţia jucătorilor în teren pe zone de influenţe şi distribuţia calităţilor în funcţie de tipul confruntărilor şi zone, putem să trecem la nucleul aplicaţiei – simularea meciului.

Această componentă, prin complexitatea ei, a necesitat o analiză atentă a tuturor posibilităţilor de simulare a meciului, fiind identificate trei modalităţi de abordare:

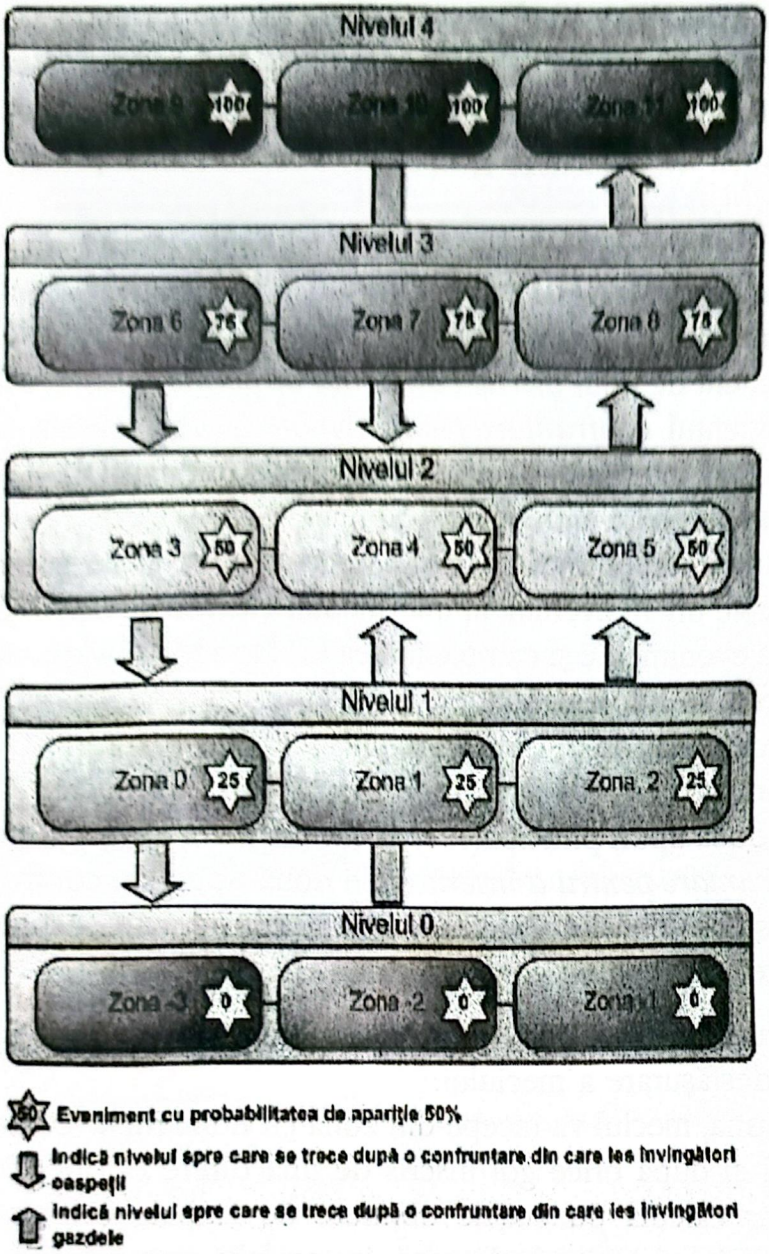
- prima variantă se bazează, în cea mai mare parte, pe inteligenţa artificială, urmând a fi create 22 de fire de execuţie (câte unul pentru fiecare jucător) care să trateze independent fiecare fază a meciului.

Jucătorii – reprezentaţi, de exemplu, de agenţi software inteligenţi – au capacitatea de a analiza fiecare fază, de a realiza schimb de mesaje cu ceilalţi jucători şi de a lua propria decizie. Această variantă este indicat să fie folosită în special dacă se doreşte o prezentare grafică a meciului.

- cea de-a două variantă ar fi simularea bazată în totalitate pe evenimente. Meciul începe dintr-un punct de start şi intră direct în evenimentul *confruntarea pentru balon*. De aici, dacă se ajunge la un eveniment de tipul *pierde balon*, *aut de poartă*, *ofsaid*, *aut* se revine la evenimentul *confruntare pentru balon*. În caz contrar, se poate trece într-o fază intermediară *câştigă balon*, de unde, printr-o *pasă*, *centrare* sau *şut* se poate ajunge într-o situaţie de gol. La fel se poate ajunge şi prin intermediul unui *corner*, *henţ* sau *fault*, dar în această situaţie se foloseşte un alt eveniment intermediar (*lovitura liberă*). Dependenţele dintre evenimente şi complexitatea lor fac această variantă să fie destul de greu de implementat.
- în fine, cea de-a treia variantă (cea despre care vom vorbi în continuare) este de asemenea bazată pe evenimente, dar e orientată spre două tipuri principale: *confruntare pentru câştigarea balonului* şi *confruntare pentru a înscrie* (cele două tipuri de confruntări amintite mai sus). Celelalte evenimente sunt considerate secundare şi vor fi tratate în partea de generare a comentariilor.

Având jucătorii distribuiţi pe zone, putem să detaliem, în continuare, modul de desfăşurare a meciului.

Întotdeauna, meciul va începe din zona cu numărul 4 (centrul terenului), ca de altfel şi după orice gol înscris de una dintre echipe. Pentru această zonă se vor calcula punctajele obţinute de ambele echipe. Echipa care obţine punctajul superior înaintează, iar cealaltă este nevoită să se retragă. Jucătorii care participă la faza respectivă sunt cei care au influenţe pe zona 4. În situaţia în care câştigă gazdele, acestea vor urca într-una dintre zonele 6, 7, 8, unde va avea loc o confruntare între fundaşii oaspeţilor şi atacanţii gazdelor; în caz contrar, vor avansa oaspeţii într-una dintre zonele 0, 1, 2 şi confruntarea următoare va fi între atacanţii oaspeţilor şi fundaşii gazdelor. De aici se poate ajunge pe unul dintre nivelurile „critice" (0 pentru gazde şi 4 pentru oaspeţi), unde vom avea o ocazie de gol, portarul echipei care se află în apărare fiind „faţă în faţă" cu un jucător advers. Dacă acesta înscrie, meciul se reia din zona 4, în caz contrar, meciul se va relua dintr-o zonă de mijloc (nivelul 2). În primul caz, reluarea meciului din zona 4 este evidentă, al doilea îl putem compara cu o degajare a portarului, în contextul fotbalului real.



Evoluţia meciului

Să vedem totuşi cum se calculează aceste punctaje (care determină evoluţia meciului), studiind funcţia de mai jos:

```
/// <summary>
/// Calculeaza punctajul obtinut de o echipa in zona
/// respectiva.
/// </summary>
/// <param name="tp">Echipa gazda/oaspete.</param>
/// <param name="iZone">Numarul zonei de teren.</param>
/// <returns>Punctajul obtinut.</returns>
public float SkillZone(TeamPlay tp, int iZone)
{
   float fNoPoints = 0; //punctajul returnat
   float fIndPoints;    //punctaj individual
```

```
    float fInf; //influenta jucatorului pe zona data
    int iPos;   //pozitia in teren(fundas/mijlocas/atacant)

    for( int i = 0; i < Config.MATCH_PLAYERS; i ++) {
       fIndPoints = 0;
       if ( tp.play[ i].infl[ iZone] != 0 &&
            tp.play[ i].posit >= 0) {
       //jucatorul se afla pe teren si are o contributie
       //in zona respectiva
          //initializare statistica
          tp.play[ i].sTotal++;
          fInf = tp.play[ i].infl[ iZone] / 100;
          iPos = ( int)tp.play[ i].posit;
          fIndPoints += Matches.skillPos[ iPos].keep
                * tp.play[i].info.Skeep;
          fIndPoints += Matches.skillPos[ iPos].tack
                          * tp.play[ i].info.Stack;
          fIndPoints += Matches.skillPos[ iPos].pass
                          * tp.play[ i].info.Spass;
          fIndPoints += Matches.skillPos[ iPos].head
                          * tp.play[ i].info.Shead;
          fIndPoints += Matches.skillPos[ iPos].cbal
                          * tp.play[ i].info.Scbal;
          fIndPoints += Matches.skillPos[ iPos].bcon
                          * tp.play[ i].info.Sbcon;
          fIndPoints += Matches.skillPos[ iPos].spi
                          * tp.play[ i].info.Sspi;
          fIndPoints += Matches.skillPos[ iPos].sho
                          * tp.play[ i].info.Ssho;
          fIndPoints += Matches.skillPos[ iPos].pint
                          * tp.play[ i].info.Spint;
          fIndPoints += Matches.skillPos[ iPos].spe
                          * tp.play[ i].info.Sspe;
          fIndPoints += Matches.skillPos[ iPos].sta
                          * tp.play[ i].info.Ssta;
          fNoPoints += fIndPoints * fInf;
       }
    }
 return fNoPoints;
 }
```

Pentru fiecare jucător care se află pe teren şi influenţează zona respectivă, se calculează un punctaj individual. Acesta este obţinut adunând toate punctajele rezultate din produsul valorii pentru fiecare calitate şi procentajul de contribuţie a acesteia pe nivelul respectiv. Suma tuturor acestor punctaje individuale formează punctajul echipei pentru zona dată.

În mod asemănător este calculat și pentru echipa adversă, dar pentru evenimentul complementar; astfel, dacă o echipă este în apărare, cealaltă este în atac. Pentru zona de mijloc, la ambele echipe se consideră, evident, confruntarea *middle-middle*.

Având aceste două valori, următorul pas este cel de a determina echipa care obține balonul în acea zonă și, în consecință, avansează la următorul nivel. Pentru aceasta se folosește o funcție bazată pe un generator de numere aleatorii:

```
public bool TwoValues(int iFirst, int iSecond)
{
   int iFirstRandom = 0;
   int iSecondRandom = 0;
   int i = 0;
   while (i < Config.RANDOM_TWO_VALUES) {
      if (base.Next( first + second) < first)
         iFirstRandom++;
      else
         iSecondRandom++;
      i++;
   }
   return not ( iFirstRandom < iSecondRandom);
}
```

Variabilele primite de funcție ca parametri sunt cele două punctaje obținute de fiecare echipă în parte. Funcția va returna *true* în cazul în care câștigă prima echipă (gazdele) și *false* în caz contrar.

Principiul de generare este simplu: se adună cele două numere, după care se generează aleatoriu un număr cuprins între 0 și suma calculată. Dacă numărul obținut este mai mic decât prima valoare, atunci câștigă prima echipă, altfel, învinge a doua.

Prin teste s-a observat că nu întotdeauna această abordare conduce la cea mai bună soluție. Să considerăm următorul exemplu: prima echipă are o valoare de 3.333 puncte, iar cea de-a doua – 6.666 puncte (dublu față de prima). La un număr de generări suficient de mare (40, de exemplu), se vor obține rezultatele așteptate, adică un raport de 1/2 (prima echipă câștigă de 1/3 ori din numărul total de confruntări, iar cea de-a doua – de 2/3 ori). Însă, având în vedere că nu întotdeauna se ajunge la un asemenea număr de faze pe acea zonă, funcției i s-a adăugat un *controller* care repetă generarea de un număr constant de ori. De asemenea, prin teste, valoarea determinată a fost 5. Altfel spus, funcția este apelată de 5 ori și câștigă echipa care ajunge prima la trei „victorii". O valoare prea mare pentru acest *controller* ar fi determinat ca în majoritatea cazurilor să câștige echipa având valoarea superioară, simularea nemaifiind în acest caz realistă.

2.5 Durata meciului

Teoretic, jocul se desfășoară pe durata a 90 de minute. Pentru fiecare fază se alocă o durată reprezentată de un număr aleatoriu cuprins între `MATCH_MIN_PHASE` (durata minimă: 30 de secunde) și `MATCH_MAX_PHASE` (durata maximă: 70 de secunde). În realitate însă, meciul este generat instantaneu, duratele fazelor fiind folosite doar la generarea comentariilor din meci pentru păstrarea corespondenței între fază și minutul jocului.

Cele două valori de mai sus au o importanță deosebită în economia meciului; utilizarea unor valori mari va determina un număr mic de faze și, în consecință, scoruri mici. Contrar, valorile mici vor implica un număr mare de faze și scoruri mari.

Meciul se desfășoară pe o perioadă de 90 * 60 = 5400 secunde. Inițial, se pornește meciul cu `iSecond = 0`, apoi la fiecare fază se adaugă numărul de secunde alocate pentru faza respectivă. Când se ajunge la `iSecond = 5400`, meciul este oprit și sunt salvate scorul înregistrat, comentariile generate pe parcurs și statisticile finale. Despre toate acestea însă vom discuta în secțiunea următoare.

2.6 Evenimentele meciului

Alături de confruntările care au loc în timpul meciului pentru câștigarea balonului, în orice moment pot apărea diverse evenimente, corespunzătoare celor din fotbalul real, cum ar fi ocazie de gol, lovitură liberă, penalti, accidentare, cartonașe și – bineînțeles – goluri. Să vedem care sunt acestea în cazul nostru:

- `MATCH_RAISE_NONE` – evenimentul nul (în cazul apariției acestui eveniment, nu are loc nici o acțiune; este utilizat deoarece în orice moment trebuie generat un eveniment);
- `MATCH_RAISE_OCCASION` – ocazie de a înscrie; acest eveniment poate fi descompus în alte două subevenimente:
  - *Gol* – echipa înscrie din acțiune;
  - *Ratare* – echipa ratează ocazia avută;
- `MATCH_RAISE_PENALTY` – echipa obține un penalti:
  - *Gol* – echipa înscrie din penalti;
  - *Ratare* – echipa ratează penaltiul;
- `MATCH_RAISE_FREEKICK` – echipa obține o lovitură liberă:
  - *Gol* – echipa înscrie din lovitură liberă;
  - *Ratare* – echipa nu reușește să înscrie din lovitură liberă;
- `MATCH_RAISE_SIMULLATION_Y` – cartonaș galben primit în urma unei simulări;

- MATCH_RAISE_FAULT_Y – cartonaş galben primit ca urmare a unui fault;
- MATCH_RAISE_NOTFAIR_Y – cartonaş galben primit în urma unui comportament nesportiv;
- MATCH_RAISE_PROTEST_Y – cartonaş galben primit în urma unor proteste;
- MATCH_RAISE_FAULT_R – cartonaş roşu primit ca urmare a unui fault dur;
- MATCH_RAISE_LAST_DEF_R – cartonaş roşu primit în urma unui fault făcut din postura de ultim apărător;
- MATCH_RAISE_ACCIDENT_REP – înlocuire ca urmare a unei accidentări;
- MATCH_RAISE_ACCIDENT – accidentare, dar jucătorul continuă meciul;
- MATCH_RAISE_REPROOF – reproşuri din partea antrenorului;
- MATCH_RAISE_SEC_YELLOW_R – cartonaş roşu obţinut în urma primirii celui de-al doilea galben.

Unul dintre aceste evenimente va fi generat, pe baza unei scheme de probabilităţi, de fiecare dată când se va schimba zona de joc. Să vedem această schemă, după care vom intra mai în detalii, pentru înţelegerea desfăşurării meciului.

	Nivelul 0	Nivelul 1	Nivelul 2	Nivelul 3	Nivelul 4
Ocazie	0	0	0	0	750
Penalti	0	0	0	0	150
Lovitură liberă	0	0	0	100	0
Cartonaş galben (simulare)	0	0	0	10	0
Cartonaş galben (fault)	0	65	10	10	0
Cartonaş galben (comportament nesportiv)	0	5	5	5	0
Cartonaş galben (proteste)	0	5	5	5	0
Cartonaş roşu (fault)	0	5	5	0	0
Cartonaş roşu (ultim apărător)	0	5	5	5	0
Înlocuire	0	5	5	5	0
Accidentare	0	5	5	5	0
Reproşuri	0	5	5	5	0
Eveniment vid	1000	900	955	855	100

Evenimentele repartizate pe niveluri de joc

Din tabelul de mai sus putem vedea ce evenimente apar pe fiecare nivel în parte. De exemplu, în zona de atac (nivelul 4) poate apărea doar o ocazie (cu o probabilitate de 75%) sau un penalti (probabilitate de 15%). Dacă echipa gazdă se află pe o zonă a nivelului 4 (este în atac), echipa oaspete va fi în apărare, şi ei îi va corespunde nivelul 0. În acest caz, prioritate pentru generarea evenimentului va avea echipa gazdă, oaspeţii având evenimentul nul. Pe celelalte niveluri pot apărea şi restul evenimentelor enumerate mai sus.

Vom vorbi în continuare însă despre comentariile din meci, urmând să revenim ulterior la evenimente, între acestea existând o strânsă legătură.

Evenimentele care au loc în cadrul meciului se pot urmări prin intermediul comentariilor generate de acest modul. Pentru fiecare tip de evenimente, există câte o categorie de comentarii care descriu situaţia respectivă prin diferite fraze. În funcţie de identificatorul evenimentului generat, va fi determinată o frază dintr-o colecţie, procesată pentru înlocuirea parametrilor, la sfârşit obţinându-se comentariul final.

Colecţia de fraze este memorată într-un fişier XML cu următorul format:

```
<?xml version="1.0" encoding="utf-8"?>
<phases xmlns="http://tempuri.org/phases.xsd">
  ...
    <phase id="01">
      <sample>
          {19} a ales o interesanta asezare tactica {2}.
      </sample>
      <sample>
          {19} a optat pentru o strategie de joc {2}.
      </sample>
      <sample>Antrenorul lui {19} a preferat o asezare
              in teren {2}.
      </sample>
    </phase>
    <phase id="02">
      <sample>De departe cel mai bun om din teren a fost
              {13}, jucatorul {12}.
      </sample>
      <sample>Cel mai important jucator din teren a fost
              fara indoiala {13}, jucatorul lui {19}.
      </sample>
      <sample>Omul meciului a fost creditat jucatorul
            echipei {19}, {13}.
      </sample>
    </phase>
  ...
</phases>
```

Acest fişier, denumit *phases.xml*, are asociată o schemă *phases.xsd*, folosită pentru validarea acestuia. Validarea efectivă este realizată prin intermediul celor două funcţii C# de mai jos:

```
public bool IsXMLValid( string sXmlFile, string sXsdFile)
{
   bIsValid = false;
   //Instantierea unui obiect de tip XmlTextReader pentru
   //citirea fisierului XML
   XmlTextReader textReader = new XmlTextReader(sXmlFile);

   //Asocierea unui validator obiectului textReader
   XmlValidatingReader validator = new
      XmlValidatingReader( textReader);
   validator.ValidationType = ValidationType.Schema;

   //Crearea unui obiect de tip XmlSchemaCollection
   //care sa contina schema
   XmlSchemaCollection schema = new XmlSchemaCollection();
   schema.Add( null, new XmlTextReader( sXsdFile));

   //Adaugarea schemei in lista schemelor pe care la va
   //folosi validatorul (in cazul nostru, una singura)
   validator.Schemas.Add( schema);

   //Adaugarea unei metode in lista de evenimente a
   //validatorului care va semnala erorile produse
   ValidationEventHandler eventHandler = new
      ValidationEventHandler( ValidationHandler);
   validator.ValidationEventHandler += eventHandler;

   //Citirea documentului (continutul nu este folosit,
   //deoarece realizam doar o validare)
   while ( validator.Read())
   {}

   return bIsValid;
}

public void ValidationHandler( object sender,
                ValidationEventArgs args)
{
   bIsValid = false;
}
```

Aceste două funcţii sunt incluse în clasa `Common`, unde `bIsValid` este o variabilă privată, care la începutul validării fişierului XML are valoarea *false*.

Dacă documentul nu este valid, validatorul va emite un eveniment, tratat de funcția `ValidationHandler`, unde `bIsValid` va deveni *false*. La sfârșitul funcției `IsXmlValid` este returnată valoarea lui `bIsValid`.

După ce am urmărit procesul de validare a documentului, să detaliem modul în care se realizează procesarea acestuia – prin intermediul metodelor oferite de mediul .NET – și cum memorăm colecția de fraze obținută.

```
private static void LoadPhases()
{
    int id = 0;
    //crearea unei instante a clasei XmlTextReader, ce
    //primeste ca parametru numele fisierului XML (phases)
    XmlTextReader reader =
       new XmlTextReader(Config.XML_PHASES);
    //citim nodurile din fisier secvential
    while (reader.Read())
    {
       if ( reader.IsStartElement("phase"))
       {
          //obtinem identificatorul frazei
          id = Convert.ToInt32(reader.GetAttribute("id"));
       }
       else if (reader.IsStartElement("sample"))
       {
          //e incrementat nr. de fraze de un anume tip
          nPhaseType[ id] ++;
       }
       else if ( reader.HasValue )
       {
          //fraza este adaugata intr-un HashTable, cheia
          //obtinandu-se printr-o functie separata
          phaseHT.Add( GetKey( id, nPhaseType[ id]),
                       reader.Value);
       }
    }
    reader.Close();
}
```

Pentru adăugarea ulterioară a unor noi evenimente, a fost alocat un număr total de 100 de identificatori, dintre care sunt folosiți aproximativ jumătate (restul urmând a fi folosiți într-o versiune ulterioară). Lista completă a acestora se găsește în tabelul următor:

Identificatorul frazei	Categoria frazei	Descriere
1	Nu are	Aşezarea tactică
2	Nu are	Omul meciului
5	Nu are	Realizare *Hattrick* (3 goluri marcate de acelaşi jucător)
6	Reproşuri	Reproşuri
8	Accidentare	Accidentare
9	Nu are	Nemulţumirea suporterilor
10	Ocazie	Deschidere scor din acţiune
11	Ocazie	Preluarea conducerii
12	Ocazie	Mărirea diferenţei
13	Ocazie	Reducere din diferenţă
14	Ocazie	Egalare
15	Ocazie	Gol fără descrierea evoluţiei scorului
20	Ocazie	Ratarea deschiderii scorului
21	Ocazie	Ratarea obţinerii avantajului
22	Ocazie	Ratarea măririi avantajului
23	Ocazie	Ratarea reducerii din diferenţă
24	Ocazie	Ratarea egalării
25	Ocazie	Ratare fără descrierea evoluţiei scorului
30	Lovitură liberă	Deschiderea scorului din lovitură liberă
31	Lovitură liberă	Preluarea conducerii din lovitură liberă
32	Lovitură liberă	Mărirea avantajului din lovitură liberă
33	Lovitură liberă	Reducerea diferenţei din lovitură liberă
34	Lovitură liberă	Egalare din lovitură liberă
35	Lovitură liberă	Egalare din lovitură liberă fără descrierea evoluţiei scorului
40	Lovitură liberă	Ratarea deschiderii scorului din lovitură liberă
41	Lovitură liberă	Ratarea obţinerii avantajului din lovitură liberă
42	Lovitură liberă	Ratarea măririi avantajului din lovitură liberă
43	Lovitură liberă	Ratarea reducerii avantajului din lovitură liberă
44	Lovitură liberă	Ratarea egalării din lovitură liberă
45	Lovitură liberă	Ratarea egalării din lovitură liberă fără descrierea evoluţiei scorului
50	Penalti	Deschiderea scorului din penalti
51	Penalti	Preluarea conducerii din penalti
52	Penalti	Mărirea avantajului din penalti
53	Penalti	Reducerea diferenţei din penalti
54	Penalti	Egalare din penalti
55	Penalti	Egalare din penalti fără descrierea evoluţiei scorului
60	Penalti	Ratarea deschiderii scorului din penalti
61	Penalti	Ratarea obţinerii avantajului din penalti
62	Penalti	Ratarea măririi avantajului din penalti
65	Penalti	Ratarea reducerii avantajului din penalti
64	Penalti	Ratarea egalării din penalti
65	Penalti	Ratarea egalării fără descrierea evoluţiei scorului
70	Cartonaş galben	Cartonaş galben pentru simulare
71	Cartonaş galben	Cartonaş galben ca urmare a unui fault
72	Cartonaş galben	Cartonaş galben ca urmare a unor proteste
73	Cartonaş galben	Cartonaş galben în urma unui comportament nesportiv
80	Cartonaş roşu	Cartonaş roşu pentru un fault dur
81	Cartonaş roşu	Cartonaş roşu obţinut după al doilea galben
82	Cartonaş roşu	Cartonaş roşu obţinut din ipostaza de ultim apărător
90	Înlocuire	Înlocuire a unui jucător

Se poate observa că fiecare categorie de fraze are alocate cel mult 10 subcategorii. Deci, dacă avem evenimentul *gol din lovitură liberă*, acesta este

împărțit în alte 6 subcategorii: deschidere scor, obținere avantaj, egalare, mărire diferență, reducere din diferență și evoluție nedescrisă a scorului.

Pentru fiecare tip au fost construite mai multe comentarii care descriu evenimentul respectiv. La generarea evenimentului, este selectat aleatoriu un identificator pentru faza corespunzătoare. Să analizăm un exemplu de fază:

Jucătorul lui {19}, {13}, a adus {10} la conducere după {0} de minute, {15}-{25} în urma unei pătrunderi de pe {17}.

Evident fazele nu sunt complete, forma lor finală obținându-se după procesare și înlocuirea parametrilor găsiți cu valorile corespunzătoare. În exemplul de mai sus, substituțiile care vor fi efectuate sunt:

{19} – numele echipei din prim-plan;
{13} – jucătorul principal (la o fază pot participa mai mulți jucători);
{10} – echipa la cazul acuzativ („gazdele" sau „oaspeții");
{0} – minutul în care are loc evenimentul;
{15} – numărul de goluri marcate de echipa care este în prim-plan;
{25} – numărul de goluri marcate de echipa adversă;
{17} – partea terenului în care are loc acțiunea.

Funcția care înlocuiește parametrii comentariului cu valorile corespunzătoare este prezentată mai jos:

```
public string RaisePhase( bool bTeam, int iPhaseID)
{
   string sPhaseBegin; //Fraza initiala
   string sPhaseEnd = string.Empty; //Fraza finala
   //Se actualizeaza parametrii cu valorile curente din meci
   UpdatePhaseData( bTeam);
   //Este obtinuta fraza din colectia existenta
   sPhaseBegin = Phases.GetPhase( iPhaseID);
   int i = 0;
   bool bOpen = false;
   string sParam = string.Empty;
   for ( i = 0; i < sPhaseBegin.Length; i++)
   {
      //Inceput de parametru
      if ( sPhaseBegin[ i] == '{')
      {
         bOpen = true;
      }
      //Sfarsit de parametru
      else if ( sPhaseBegin[ i] == '}')
      {
         bOpen = false;
         int iPos = sPhaseEnd.Length;
         string data =
            dataPhase[ Convert.ToInt32(sParam)];
```

```
            if ( iPos == 0)
               //Prima litera este majuscula
               sPhaseEnd += data[0].ToString().ToUpper() +
                            data.Remove(0, 1);
            else
               sPhaseEnd += data;
            sParam = string.Empty;
         }
         else
         {
            if ( bOpen)
               sParam += sPhaseBegin[ i];
            else
               sPhaseEnd += sPhaseBegin[ i];
         }
      }
   }
```

Un comentariu final, după această procesare, ar putea fi: *Jucătorul lui Keiser Sose, Daniel Puiu, a adus gazdele la conducere după 25 de minute, 2-1 în urma unei pătrunderi de pe stânga.*

Lista completă de parametri poate fi parcursă în tabelul de mai jos:

ID	Descriere
0	Minutul de joc
1	Numărul de minute de la faza precedentă
10	Echipa la cazul nominativ („gazdele"/„oaspeţii") - prima echipă
11	Echipa la cazul acuzativ („gazde"/„oaspeţi") - prima echipă
12	Echipa la cazul genitiv („gazdelor"/„oaspeţilor") - prima echipă
13	Jucătorul principal (prima echipă)
14	Jucătorul secundar (prima echipă)
15	Numărul de goluri înscrise de prima echipă
16	Partea terenului în care are loc faza în cazul unei acţiuni desfăşurate pe flancuri („stângă"/„dreaptă") - prima echipă
17	Partea terenului pe care are loc faza în cazul unei acţiuni („stângă"/„centru"/„dreaptă") - prima echipă
19	Numele echipei - prima echipă
20	Echipa la cazul nominativ („gazdele"/„oaspeţii") - a doua echipă
21	Echipa la cazul acuzativ („gazde"/„oaspeţi") - a doua echipă
22	Echipa la cazul genitiv („gazdelor"/„oaspeţilor") - a doua echipă
23	Jucătorul principal (a doua echipă)
24	Jucătorul secundar (a doua echipă)
25	Numărul de goluri înscrise de a doua echipă
26	Partea terenului pe care are loc faza în cazul unei acţiuni desfăşurate pe flancuri („stângă"/„dreaptă") - a doua echipă
27	Partea terenului în care are loc faza în cazul unei acţiuni („stângă"/„centru"/„dreaptă") - a doua echipă
29	Numele echipei - a doua echipă

Parametrii sunt actualizaţi continuu, astfel încât aceştia să conţină întotdeauna ultimele informaţii din meci. Şi în acest caz au fost alocaţi mai mulţi identificatori (30), în cazul adăugării unora noi.

Din tabelul precedent se poate observa că evenimentelor corespunzătoare golurilor sau ratărilor le corespund mai multe tipuri de fraze. Dacă am fi avut un singur tip de comentariu, nu am fi putut să urmărim evoluţia scorului.

Să considerăm un exemplu. Presupunem că s-a generat evenimentul *Gol din acţiune*. În mod implicit, am fi avut următoarele comentarii:

- *Câteva manevre precise pe {17} au dus în minutul {0} la golul înscris de {13}, care a schimbat tabela la {15}-{25}.*
- *{13} s-a înălţat mult peste adversari când a înscris cu capul pentru {15}-{25}, urmare a unui corner după {0} de minute de joc.*
- *În cel de-al {0}-lea minut al meciului, apărarea centrală a oaspeţilor nu a putut face altceva decât să privească impetuoasa cursă a lui {13}, care a marcat pentru {11} golul de {15}-{25}.*
- *În urma unei acţiuni a echipei {11}, {13} a recepţionat un balon ce a trecut prin întreaga linie defensivă a {22}, trimiţându-l uşor pe lângă portar în minutul {0} pentru {15}-{25}.*
- *{13} s-a înălţat mult peste adversari când a înscris cu capul pentru {15}-{25}, urmare a unui corner după {0} de minute de joc.*
- *{13} de la {11} a făcut o cursă impresionantă, depăşindu-l pe lentul {24} şi încheind rapidul atac cu golul de {15}-{25}.*
- *{14} a ajuns primul la balon după lovitura de colţ, dar lovitura sa cu capul a fost respinsă de portar. Mingea a ricoşat însă la alt jucător al lui {19}, {14}, care a reuşit să înscrie golul de {15}-{25}.*

Studiind frazele date, se poate remarca faptul că fiecare dintre acestea conţine doar scorul, neoferind însă detalii despre evoluţia sa. Pentru a înlătura acest deficit, fiecărui eveniment de gol (acţiune, penalti, lovitură liberă) i se asociază câte 5 tipuri de fraze:

- Deschiderea scorului

{10} au deschis scorul *în minutul {0} cu {15}-{25}, în urma unor acţiuni de mare fineţe pe centrul terenului. Autorul golului a fost {13}.*

- Obţinerea avantajului

După {1} minute, {19} a luat conducerea cu *{15}-{25}, în urma unei acţiuni solitare a lui {13}, care a pătruns în careu de pe extrema {16} şi a trimis balonul peste portarul {22}.*

- Mărirea avantajului

Puţin mai târziu, o combinaţie magnifică pe {17} a făcut {19} să-şi mărească avantajul *prin {13} la {15}-{25}.*

- Reducerea din diferenţă

{19} a redus scorul *la {15}-{25}, când {13} a nimerit ţinta după o bună lansare pornită de pe extrema {16}, în minutul {0}.*

- Egalarea

În cel de-al {0}-lea minut al partidei, {19} a reuşit egalarea *{15}-{25}. {13} a preluat o minge venită de pe dreapta şi a deviat-o pe lângă portarul advers.*

Pentru selectarea frazei corespunzătoare evenimentului, este generat aleatoriu un număr cuprins între 0 şi numărul total de fraze din categoria evenimentului respectiv. În mod similar, e selectată şi categoria comentariului. Nu întotdeauna va fi însă afişată o frază care să descrie evoluţia scorului. Au fost setate două variabile, prima cu 60 şi a doua cu 40. Astfel, sunt 60% şanse ca fraza respectivă să nu conţină descrierea evoluţiei scorului şi 40% şanse, în celălalt caz.

În continuare, vom analiza două dintre funcţiile care au un rol esenţial în simularea meciului. Vom începe cu funcţia `MatchSimulator()` care este, evident, responsabilă cu simularea meciului:

```
public void MatchSimulator()
{
int homes = 0, aways = 0; //Nr. de ocazii gazde/oaspeti
bool team; //Identificator pentru echipa
int phase; //Numarul de secunde pentru o faza de meci
bool pause = true; //Indica pauza meciului
bool goal = false; //Are valoare true daca echipa inscrie
scorA = 0; //Numarul de goluri inscrise de oaspeti
scorH = 0; //Numarul de goluri inscrise de gazde
int eventt; //ID-ul evenimentului generat
int level; //Nivelul curent
bool teamPhase; //Indica echipa care genereaza faza

//Meciul incepe din zona 4
zone = 4;
//Este generata partea de inceput a fisierului xml care va
//contine „filmul meciului"
InitMovie();
//Meciul a inceput
second = 0;
while (second < Config.MATCH_TIME)
{
   //Este obtinut nivelul curent
   level = Phases.GetLevel(zone);

   //Se determina care echipa va genera comentariul
   //true - echipa gazda / false - echipa oaspete
   teamPhase = Phases.ChooseLevel( level);
   if (!teamPhase)
      //E obtinut nivelul complementar pt. echipa adversa
      level = Phases.InverseLevel( level);
```

```
//Determinam evenimentul care apare in zona data
eventt = Phases.GetEvent( level);

//Este generat evenimentul determinat
RaiseEvent( eventt, teamPhase, zone);
if ( goal)
   zone = 4;
else if ( zone <= 8 && zone >= 0) {
   //Are loc o confruntare pentru castigarea balonului
   team = rand.TwoValues( (int)SkillZone( this.ho,
            zone), (int)SkillZone( this.aw, zone));
    //Variabile utilizate pentru statistica

   ho.sTotal ++;
   aw.sTotal ++;
   if ( team) { //A castigat prima echipa
      //Initializare statistica gazde
      for ( int i = 0; i < Config.MATCH_PLAYERS; i++)
         if ( ho.play[i].infl[zone] != 0 &&
               ho.play[i].posit >= 0) {
            ho.play[i].sWin ++;
         }
      ho.sWin ++;
      //Gazdele avanseaza
      zone = (zone / 3) * 3 + 3 + rand.Next( 3);
   }
   else {
      //Initializare statistica oaspeti
      for ( int i = 0; i < Config.MATCH_PLAYERS; i++)
         if ( aw.play[i].infl[zone] != 0 &&
               aw.play[i].posit >= 0)
         {
            aw.play[i].sWin ++;
         }
         aw.sWin ++;
         //Oaspetii avanseaza
         zone = (zone / 3) * 3 - 3 + rand.Next( 3);
   }
else {
   if (zone < 0)
      aways++;//Oaspetii au avut o ocazie dar nu
              //au inscris
   else
      homes++;//Gazdele au avut o ocazie dar nu
              //au inscris
   //Meciul este reluat de la mijlocul terenului
```

```
            int[] middle = {1, 1, 1};
            zone = 3 + rand.NValues( middle.Length, middle);
        }
        //Pauza meciului (jocul este oprit)
        if ( second >= Config.MATCH_TIME / 2 && pause) {
            pause = false;
        }

    //Se determina numarul de secunde pentru faza respectiva
    phase = rand.Next( Config.MATCH_MIN_PHASE,
                       Config.MATCH_MAX_PHASE);
    second += phase;
    }

    //Se salveaza meciul in fisierul xml
    FinishMovie();

    //Sunt actualizate rezultatele in tabela matches
    Bridge.UpdateMatch( this.id,this.movie, this.statistics,
    this.scorH, this.scorA);
    Bridge.UpdateMovie (this.id, this.movie);
    Bridge.UpdateMatchStatus( this.id,
                    Config.MATCH_STATUS_LIVE);
    //Sunt actualizate informatiile despre jucatori in tabela
    //players (cartonase, goluri etc.)
    for ( int i = 0; i < Config.MATCH_LOT; i++) {
        Bridge.UpdatePlayer( ho.play[ i].info.Id,
            ho.play[ i].info.Ycard + ho.play[ i].nYellow,
            ho.play[ i].info.Rcard + ho.play[ i].nRred,
            ho.play[ i].info.Goals + ho.play[ i].mGoals);
        Bridge.UpdatePlayer( aw.play[ i].info.Id,
            aw.play[ i].info.Ycard + aw.play[ i].nYellow,
            aw.play[ i].info.Rcard + aw.play[ i].nRred,
            aw.play[ i].info.Goals + aw.play[ i].mGoals);
    }
    }
```

Dacă funcția descrisă mai sus are rolul de a genera meciul în ansamblul său (evoluția timpului, determinarea echipei câștigătoare în urma unei confruntări pentru balon, generarea evenimentelor, actualizarea informațiilor din teren și chiar salvarea acestora în baza de date), `RaiseEvent()` tratează evenimentele generate.

```
/// <summary>
/// Functie de o importanta deosebita:
/// construieste evenimentele din meci.
```

```
/// </summary>
/// <param name="eventID">Identificatorul
/// evenimentului.</param>
/// <param name="teamPhase">Echipa care detine
/// evenimnetul (true=gazde).</param>
/// <param name="zone">Zona terenului.</param>
/// <returns>Returneaza identificatorul fazei.</returns>
public int RaiseEvent ( int eventID, bool teamPhase,
                        int zone)
{
 goal = false;
 int phaseID = -1;
 if ( teamPhase)//gazdele au ocazia
 {
   switch( eventID) {
     case Config.MATCH_RAISE_SEC_YELLOW_R:
       phaseID = SelectPhase(
         Config.MATCH_RAISE_SEC_YELLOW_R, goal,
         scorH, scorA);
       this.ho.play[ hPlayer1].posit = Position.OUT;
       RedistributePlayers( ho);
       break;
     case Config.MATCH_RAISE_OCCASION:
       hPlayer1 = GetScoringPlayer( teamPhase);
       hPlayer2 = GetPlayer( team-Phase, zone - 6);
       aPlayer1 = GetKeeper( !team-Phase);
       aPlayer2 = GetPlayer( !team-Phase, zone - 6);
       goal = Scoring
         (Cofig.MATCHES_PLAYERS_SCORING_ACTION,
          Config.MATCHES_PLAYERS_KEEPING_ACTION,
          teamPhase, hPlayer1, aPlayer1);
       phaseID = SelectPhase(
          Config.MATCH_RAISE_OCCASION, goal, scorH, scorA);
       if ( goal) {
         scorH ++;
         this.ho.play[ hPlayer1].mGoals ++;
       }
       else {
         this.ho.sOccasion ++;
       }
       //RaisePhase( teamPhase, phaseID);
       break;
   ...
   }
 }
}
```

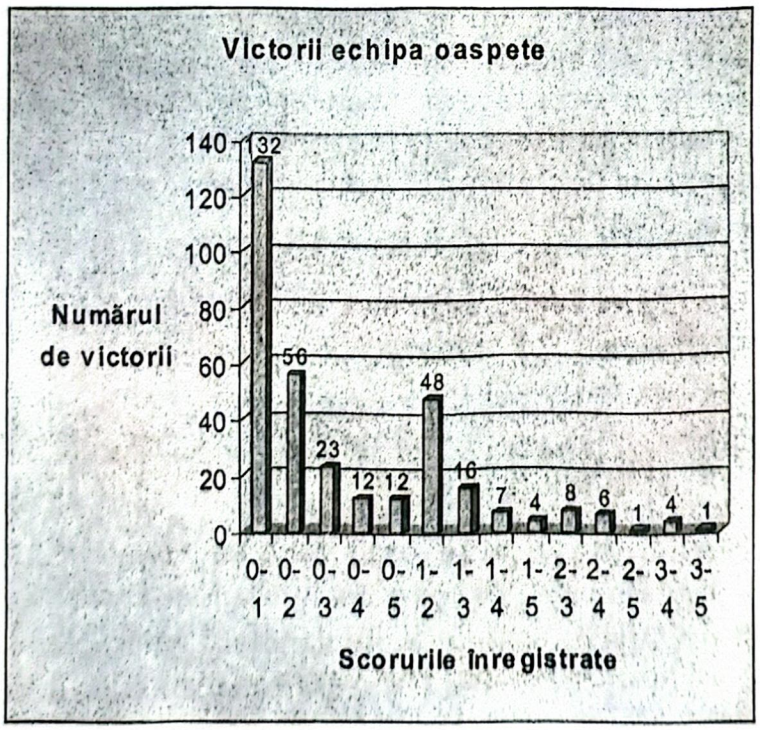
Dintre evenimentele enumerate într-un subcapitol anterior, în funcţia de mai sus au fost prezentate doar două, celelalte asemănându-se într-o oarecare măsură cu cele descrise. Să vedem cum este tratat evenimentul `MATCH_RAISE_SEC_YELLOW_R` (primirea celui de-al doilea galben, şi implicit a cartonaşului roşu):

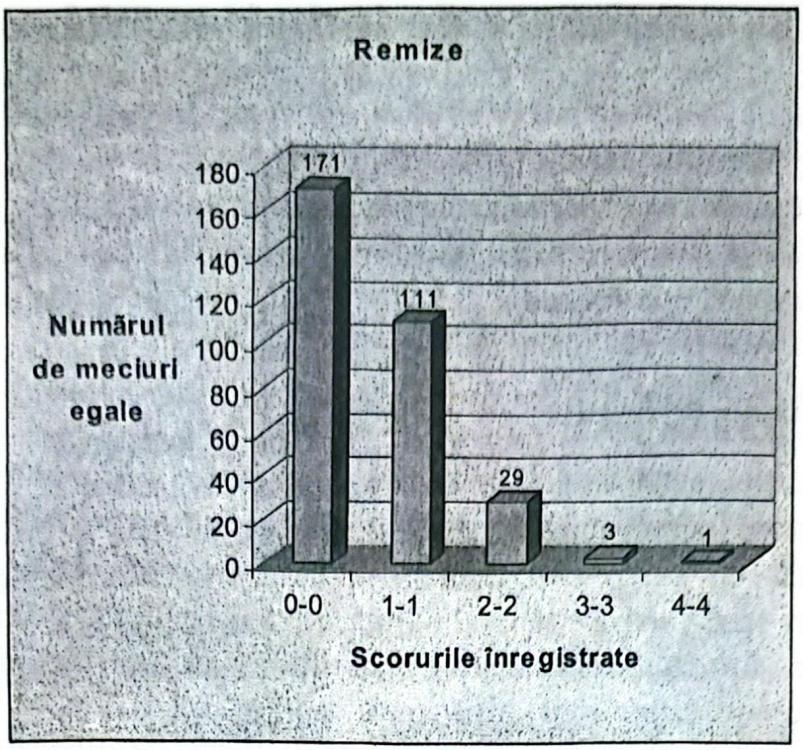
- este selectată fraza care va fi asociată ca un comentariu situaţiei respective;
- are loc o redistribuire a jucătorilor în teren, în urma acestei eliminări.

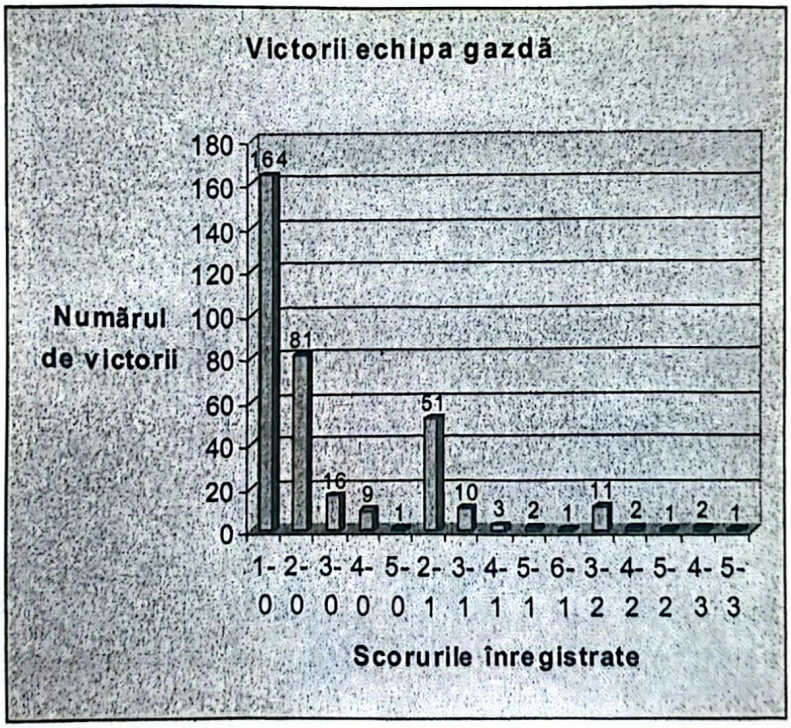
Pentru `MATCH_RAISE_OCCASION` (ocazie de gol) se procedează astfel:

- este determinat jucătorul care va încerca să înscrie, portarul cu care se va confrunta şi eventualii jucători secundari care vor participa la fază;
- este apelată funcţia `Scoring` (care se bazează pe acelaşi principiu ca şi funcţia `SkillZone()`, descrisă într-o secţiune anterioară);
- se generează comentariului în urma înscrierii unui gol sau ca urmare a unei ratări.

În încheirea acestei secţiuni, vom prezenta rezultatele obţinute prin simularea a 1.000 de meciuri între două echipe de valori absolut egale şi care au folosit formula de joc 4-4-2.







Statistici ale rezultatelor obținute în urma simulărilor meciurilor

Din figurile de mai sus se poate observa repartiția scorurilor înregistrate. Egalitatea dintre cele două echipe este evidentă, scorurile egale sau apropiate fiind dominante.

În subcapitolul următor, simularea meciului de fotbal va fi integrată într-o aplicație Web, și anume un manager de fotbal *on-line*.

3. *Hattrick* – manager de fotbal pe Web

Managerul descris în continuare a avut ca punct de plecare unul dintre cele mai cunoscute jocuri de acest profil: *Hattrick* (*www.hattrick.org*). Design-ul sitului nostru este în mare parte asemănător cu al acestuia, având în vedere că scopul era de a simula un meci de fotbal pe Web, ceea ce s-a și reușit. Cum? Vom vedea în continuare.

3.1 Arhitectura aplicației

În cadrul aplicației noastre s-a încercat implementarea principalelor module prezente în orice manager de fotbal (meciuri, antrenamente, transferuri, căutări de jucători, personalul tehnic, stadion, finanțe). Între acestea se detașeaza, din punct de vedere al complexității, generarea meciului. Se poate afirma că întreaga aplicație este dependentă de acest modul. Chiar și baza de date a fost organizată după acest model.

Aplicația este structurată în patru module (proiecte), care împreună formează *Solution Hattrick* (proiectul Visual Studio.NET în ansamblul său):

- *Hattrick* – proiect de tipul *ClassLibrary* (în format *.dll*), reprezentând nucleul aplicației.
- *eManager* – proiect *ASP.NET Web Application*, care reprezintă proiectul Web propriu-zis.
- *HattrickManager* – proiect *WindowsApplication*, care are rolul de a administra aplicația.
- *SqlXmlClient* – aplicație de tip *WindowsApplication*, care folosește serviciile Web pentru transferul datelor între aplicație și serverul Web.

Să detaliem în continuare ce conține fiecare modul în parte.

În *Hattrick* se găsesc fișierele principale ale aplicației, clase care sunt puse și la dispoziția celorlalte module:

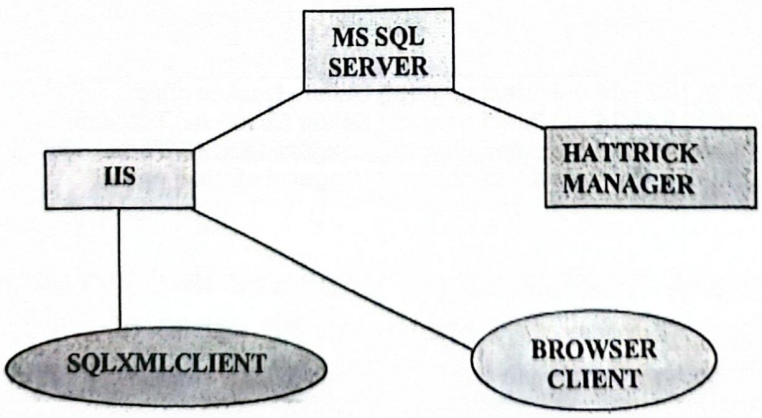
- *Bridge.cs* (clasa *Bridge*) – conține toate funcțiile care comunică cu serverul de baze de date. Cu toate că majoritatea funcțiilor din modulul *eManager* sunt implementate direct în acesta, cele care accesează baza de date se găsesc în *Bridge.cs*. Pentru realizarea conexiunii, se folosește o funcție statică ce este apelată automat o singură dată înaintea primei utilizări a funcțiilor din *Bridge*.
- *Config.cs* – conține variabile folosite pentru configurare și constantele aplicației. Cele mai importante opțiuni de configurare (date de generare, timpii alocați generării frazelor, șansele jucătorilor de a marca etc.) au fost însă stocate într-un fișier de configurare separat, pentru a permite efectuarea acestor modificări fără o nouă compilare a proiectului.

- *Hatrand.cs* – include clasa care oferă funcțiile de generare a numerelor aleatorii. A fost implementată o nouă clasă pentru a preveni generarea de numere identice pornind de la aceeași valoare.
- *Common.cs* – după numele clasei, aici se găsesc funcții folosite în mai multe module. În general, avem funcțiile care transformă valori (numere întregi) în reprezentările corespunzătoare (imagini, descrierile calităților).
- *Team.cs* – clase pentru stocarea informațiilor despre club: *UserData* (informații despre utilizatori), *TeamData* (informații despre club) etc.
- *Player.cs* – oferă accesul la toate datele despre jucători; în plus, se găsesc și funcțiile care creează efectiv jucătorii.
- *Matches.cs* – prin intermediul clasei *Matches* au loc toate inițializările premergătoare simulării meciurilor.
- *Match.cs* – conține clasa *Match*, în care avem toate funcțiile pentru generarea unui meci.

În cadrul soluției VS.NET, proiectul *Hattrick* este compilat și transformat într-o bibliotecă (DLL) ce va fi accesată din celelalte module. După ce am văzut organizarea pe module de implementare a aplicației, să discutăm în continuare despre componentele funcționale ale acesteia.

Pe lângă serverul de baze de date și serverul Web, aplicația mai conține un *daemon* (*HattrickManager*), program care rulează în fundal, având rolul de a actualiza în baza de date toate modificările care au loc în cadrul aplicației. *Daemon*-ul va comunica numai cu serverul de baze de date, care, la rândul său, va fi în legătură cu serverul Web.

De asemenea, utilizatorii au posibilitatea folosirii, alături de clientul Web obișnuit (*browser*-ul), un al doilea client (aplicație executabilă), care oferă diverse facilități pentru determinarea celor mai buni jucători sau afișarea unor statistici. Prin intermediul acestei aplicații executabile nu se poate însă administra echipa.



Interacțiunile dintre diversele componente ale proiectului

De menţionat că ambii clienţi – atât navigatorul, cât şi *SqlXMLClient* – vor comunica doar cu serverul Web *Microsoft IIS (Internet Information Services)*.

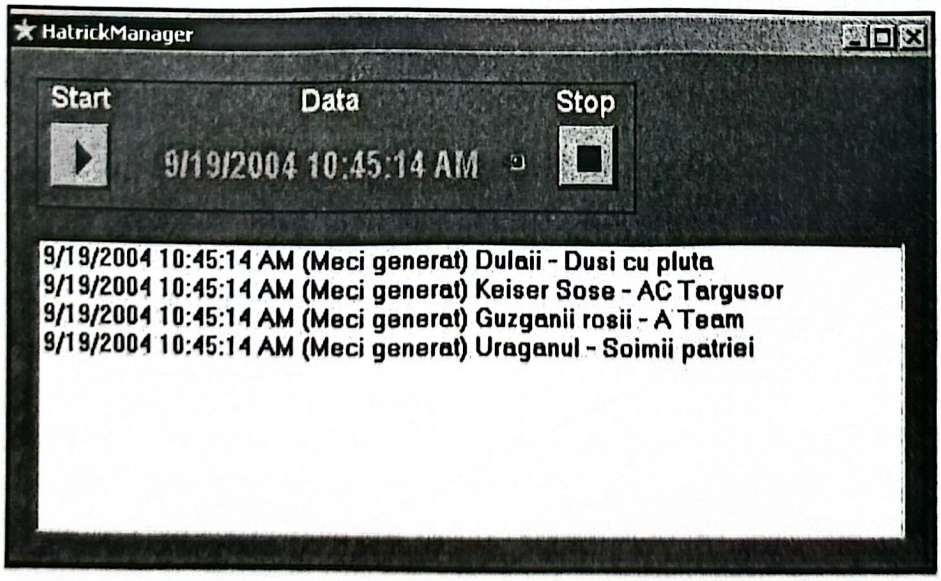
3.2 *Daemon*-ul *HattrickManager*

Să revenim acum la *HattrickManager*. Acesta poate fi considerat „inima aplicaţiei", deoarece realizează toate operaţiile care nu au legătură cu clienţii. Principala sa sarcină este, în primul rând, simularea meciurilor (despre care se va discuta ulterior). După aceasta, urmează actualizarea datelor statistice (rezultate, marcatori, goluri, cartonaşe, stări ale meciurilor etc.).

Serverul rulează continuu şi, la anumite momente setate în cadrul programului, va efectua operaţiile corespunzătoare. Interfaţa grafică este una minimală, conţinând două butoane: *Start* şi *Stop*. În mod normal, serverul va fi întotdeauna pornit, însă, în anumite cazuri speciale (întreţinerea bazei de date, de exemplu), acesta poate fi temporar oprit, în mod explicit.

La pornirea acestuia este creat un nou fir de execuţie, care va intra într-o buclă infinită, verificând de fiecare dată timpul curent cu datele de planificare a operaţiunilor. În cazul în care acestea coincid, se vor efectua procesările respective, afişându-se în acelaşi timp şi mesajele corespunzătoare.

Meciurile vor începe în fiecare sâmbătă a săptămânii, la ora 11:00 (cine are o echipă pe *hattrick.org* cunoaşte foarte bine această oră). Generarea meciurilor va avea loc însă cu 15 minute mai devreme, din motive de performanţă. Dacă pentru fiecare meci ar fi fost alocat câte un fir de execuţie separat, la un număr mare de utilizatori, serverul nu ar fi reuşit să realizeze procesările în timp util. Un alt motiv este numărul mare de accesări care ar fi trebuit făcute spre baza de date.



Interfaţa aplicaţiei *HattrickManager*

Pentru a înlătura această situaţie, meciurile sunt simulate mai devreme, iar rezultatele obţinute sunt memorate într-un câmp special în tabela *Matches*. Situaţia meciurilor însă rămâne neschimbată, şi anume – *meci nejucat*. Când se ajunge la ora stabilită pentru începerea partidelor, atunci este modificată situaţia meciurilor în *meci live*.

Astfel, utilizatorul va avea posibilitatea de a urmări *live* meciul.

La sfârşitul generărilor, având în vedere că serverul este într-o buclă infinită, acesta va continua să selecteze alte meciuri spre a fi simulate. Meciurile recent generate nu au situaţia modificată (sunt în continuare în starea de meci nejucat). Pentru a limita accesul la baza de date, se folosesc anumite variabile de control. Acestea împiedică serverul să mai iniţieze accesări inutile. Valoarea acestor variabile este resetată la o dată prestabilită.

După cum menţionam mai sus, pe lângă generarea meciurilor, *HattrickManager* mai are şi alte atribuţii, una dintre acestea fiind actualizarea tuturor informaţiilor legate de meciurile desfăşurate; aici ne referim la clasament, goluri, puncte, marcatori, cartonaşe.

Înainte de a trece la clientul Web propiu-zis, să descriem, pe scurt, formularele Web construite în ASP.NET.

3.3 ASP.NET – formulare Web

Pentru a prezenta informaţii în navigatorul client, folosim formularele Web ASP.NET (*WebForms*), care oferă o abstractizare a modelului de programare, un model orientat pe obiect şi bazat pe evenimente. Acest mediu de lucru beneficiază de toate facilităţile oferite de platforma .NET (siguranţa tipurilor, mediu de execuţie controlat, moştenire etc.) şi reprezintă o înlocuire a clasicelor formulare HTML.

În continuare, folosim denumirea de „pagină *WebForms*" pentru un formular Web din ASP.NET. O pagină *WebForms* este formată din două părţi: componenta vizuală (furnizând interfaţa cu utilizatorul) şi logica aplicaţiei (oferind implementarea efectivă). Componenta vizuală este reprezentată de un fişier cu extensia *.aspx* – acţionând ca un container pentru HTML, text static şi controale server care pot fi afişate în *browser* –, iar logica aplicaţiei este reprezentată de un fişier cu extensia *.aspx.cs* (pentru limbajul C#) sau *.aspx.vb* (pentru Visual Basic.NET). Desigur, pot fi adoptate şi alte limbaje .NET.

Fişierele *.aspx* mai sunt referite ca pagini ASP.NET. Această tehnică de separare a codului de partea de prezentare este numită „*code-behind programming*", metodă care elimină disfuncţionalităţi de programare, oferind robusteţe şi dezvoltare uşoară a paginilor.

Codul aplicaţiei (implementând logica acesteia) reprezintă, de fapt, o clasă care extinde funcţionalitatea clasei *System.Web.UI.Page*. Această clasă conţine:

- metode care tratează diferite evenimente (de exemplu, *Page_Load*, care este apelată la fiecare accesare a unei pagini Web);
- proprietăți (ca *IsPostBack* – ne semnalează dacă pagina este accesată în urma recepționării de date la server prin metoda POST);
- atribute corespunzătoare unor controale din pagina *WebForms*;
- alte date membre necesare implementării aplicației.

În ASP.NET, cu puține modificări, oricare pagină *WebForms* poate fi reutilizată în altă pagină ca un control pe partea de server, numit, prin extensie, „*user control*". Toate fișierele aplicației logice – codul din spate – sunt compilate într-o bibliotecă dinamică *.dll*, care conține o clasă derivată din *System.Web.UI.Page*. Pentru fișierele *.aspx*, la prima cerere a navigatorului, *runtime*-ul ASP.NET generează o a doua bibliotecă dinamică *.dll*, temporară, care conține o clasă .NET. Această clasă moștenește clasa din primul *.dll*, iar în momentul rulării (execuției), când un utilizator face o cerere a unei pagini, se execută *.dll*-urile asociate și rezultatul execuției este trimis la client într-unul dintre limbajele HTML, XML, WML și ECMAScript (JScript, JavaScript). Acest rezultat este influențat de facilitățile pe care le oferă navigatorul client, care poate fi chiar un dispozitiv mobil; astfel, paginile *WebForms* transmit cod HTML compatibil cu *browser*-ul.

Arhitectura ASP.NET suportă și un model asemănător vechiului *Active Server Pages* (ASP), în care elementele de interfață și codul se află în același fișier, funcționalitatea fiind aceeași cu cea oferită de modelul cu două fișiere. Totuși, există mici diferențe: la compilare, codul din pagină nu mai este compilat într-un fișier separat, clasa din *.dll*-ul asociat paginii *.aspx* derivă direct din clasa *System.Web.UI.Page*. Acest model are avantajul că poate fi facil de utilizat în dezvoltarea paginilor Web.

Unul dintre aspectele care vor duce la o dezvoltare rapidă a interfețelor utilizator pentru paginile *WebForms* îl reprezintă controalele server. Controalele server sunt folosite pentru a crea interfața utilizator în paginile *WebForms*. Caracteristicile acestora pot fi rezumate astfel: expun un model orientat pe obiect, încapsulează un mecanism de persistență a stărilor între *round-trip*-uri, generează evenimente de la client care sunt tratate la server, iar la execuție generează cod compatibil cu *browser*-ul sau cu dispozitivul destinat (detectat în mod automat).

Există două tipuri de bază în care pot fi clasificate controalele:

- *HTML Controls*: reprezintă elemente HTML care pot fi programate la nivelul serverului și expun un model obiectual restricționat la caracteristicile elementelor HTML clasice pe care le afișează;
- *Web Controls*: aduc facilități superioare controalelor HTML, incluzând controale mult mai complexe, cum ar fi cel de tip calendar, iar modelul obiectual nu reflectă neapărat sintaxa HTML.

Primul set de controale, *HTML Controls*, reprezintă elemente HTML care conţin atribute făcând posibilă programarea lor pe server. Fiecare element HTML poate fi programat pe server prin adăugarea atributului `runat="server"` şi stabilirea unui atribut ID pentru a putea fi referit controlul ca membru de clasă în codul de pe server. ASP.NET furnizează controale HTML predefinite, de la formulare până la controale pentru introducerea de informaţii text pe una sau mai multe linii, de la *checkbox*-uri până la controale de selecţie, butoane de tip *submit*, *list box*-uri, tabele, imagini şi alte controale corespunzătoare elementelor HTML.

Pentru a abstractiza mai mult modelul de programare al paginilor *WebForms*, ASP.NET defineşte un al doilea tip de controale – *Web Controls*. Numite şi controale inteligente, acestea au următoarele caracteristici:

- oferă un bogat şi consistent model obiectual de programare;
- detectează automat tipul navigatorului, iar afişarea la client va fi optimizată în funcţie de caracteristicile acestuia;
- pentru unele controale se pot defini şabloane (*template*-uri) de afişare;
- posibilitatea de a controla generarea evenimentelor pe server;
- posibilitatea de a trimite evenimente unui container din interiorul acestuia (de exemplu, un control de tip buton în interiorul unui tabel);
- legarea la surse de date a tuturor proprietăţilor controalelor, pentru a influenţa afişarea la momentul execuţiei.

Aceste controale sunt definite în spaţiul de nume *System.Web.UI.WebControls* şi moştenesc, direct sau indirect, clasa de bază *WebControl*. Adăugarea controalelor *Web Controls* în paginile *WebForms* se realizează în acelaşi mod în care se inserează un element HTML.

Vom prezenta, în continuare, unul dintre controalele cel mai des utilizate în crearea paginilor.

Repeater reprezintă o simplă listă conectată la o sursă de date care foloseşte şabloane pentru afişarea informaţiilor. Fără un şablon predefinit, programatorul este nevoit să-şi definească propriul model de afişare folosind elemente HTML. În schimb, controlul este singurul care oferă libertate maximă în specificarea modului de afişare, al stilului şi al formatării. Iată şi un exemplu, care va genera o listă numerotată de nume de persoane, fiecare având asociată o legătură spre adresa de *e-mail* a acesteia:

```
<asp:Repeater runat="server" id="lista">
<HeaderTemplate> <ol> </HeaderTemplate>
<ItemTemplate>
<li>
<a href='mailto:
   <%#DataBinder.Eval(Container.DataItem, "Email") %>'>
  <%# DataBinder.Eval(Container.DataItem, "Nume")%>
```

```
</a>
</li>
</ItemTemplate>
<FooterTemplate> </ol> </FooterTemplate>
</asp:Repeater>
```

3.4 eManager – client Web

Dacă *HattrickManager* avea rolul de a realiza managementul operațiunilor și de a actualiza datele, pagina Web corespunzătoare aplicației reprezintă legătura între utilizator și clubul său.

Pentru implementare, s-a folosit tehnologia ASP.NET pentru asigurarea omogenității cu celelalte proiecte. De asemenea, modularizarea aplicației a permis folosirea structurilor și funcțiilor utilizate în cadrul modulului *Hattrick*.

Pagina Web principală, numită *default.aspx*, a fost structurată în trei cadre interne (*iframe*-uri), fiecare având funcționalități diferite:

- *ifrmTree* – conține meniul aplicației. Acesta folosește o componentă implementată în JavaScript (*COOLjsTree* – descărcată de la adresa *http://javascript.cooldev.com/scripts/cooltree*). În modul utilizator neconectat, este încărcat un meniu cu diverse opțiuni de accesare a informațiilor generale din *Hattrick*, în celălalt caz, managerul având acces la toate facilitățile aplicației.

Meniul clientului

- *ifrmCenter* – reprezintă *frame*-ul principal *Hattrick*, aici încărcându-se fiecare pagină care va fi afișată. Frecvența de utilizare a cadrelor la construcția paginilor Web a scăzut în ultimul timp, însă acestea au câteva avantaje care nu pot fi neglijate: afișarea unui singur nume de pagină în timpul navigării, traficul redus (se va aduce pe client doar o parte din informațiile regăsite la un moment dat pe pagină), permisiunea utilizării *refresh*-ului automat etc.

- *ifrmMain* – este o parte distinctă din întreaga arhitectură a paginilor componente, în acest cadru intern regăsindu-se informațiile cele mai des actualizate: numărul de utilizatori, numărul de manageri, data de pe server (acesta este timpul care va determina efectuarea operațiilor principale) și ce se întâmplă în momentul respectiv pe *Hattrick*.

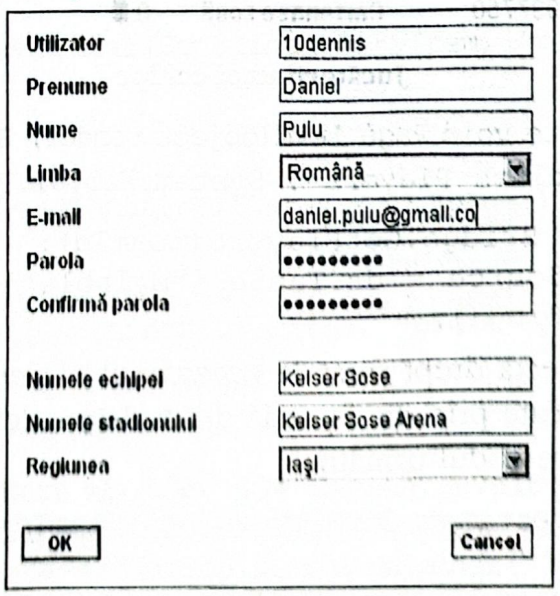
UTILIZATORI		PROGRAM	DATA SERVERULUI
Inscrişi	10	Meciuri LIVE in Divizia A !	5/25/2004
Manageri	8		2:19 AM

Rezultatele *live*

Modulul de înregistrare

Vom începe descrierea implementării efective a clientului Web cu detalierea modulului de înregistrare. Atunci când un nou utilizator, manager, în cazul nostru, dorește să se înscrie la *Hattrick*, poate alege opțiunea *Înregistrare* din meniu. Identificarea utilizatorului se realizează prin numele de cont și parolă, alese de el însuși. O dată înregistrat, utilizatorul nu devine automat activ, trebuind să aștepte înregistrarea altor utilizatori, astfel încât serverul să poată crea o nouă grupă, iar pentru aceasta are nevoie de un număr prestabilit de utilizatori (în versiunea curentă este vorba de opt utilizatori). Actualmente, acest modul înregistrează datele utilizatorului, însă acestuia nu îi este atribuită nici o echipă, divizie sau grupă, acestea urmând să-i fie asociate în momentul în care se va strânge un număr suficient de utilizatori pentru crearea unei grupe.

Formularul de înregistrare este unul clasic, în care se solicită unele informații generale despre utilizator:

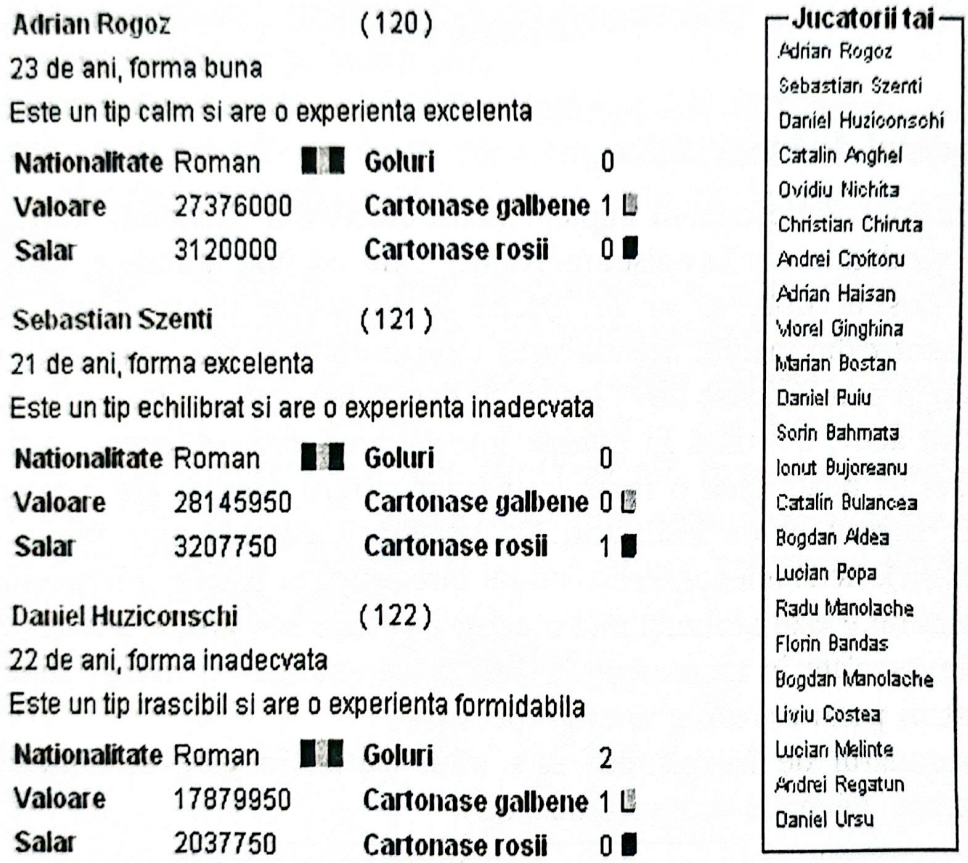


Modulul de înregistrare

Vizualizarea unei echipe

Din moment ce v-aţi înregistrat, vă puteţi conecta pentru a vizualiza echipa ce v-a fost atribuită. Expandând meniul *Club*, puteţi vizualiza informaţii despre echipă, jucători, meciuri, finanţe, stadion. În pagina *Echipă*, puteţi vizualiza informaţii despre echipă şi despre managerul echipei.

În pagina *Jucători* putem vizualiza jucătorii echipei, pentru aceasta utilizându-se un control de tip `<asp:repeater id="players" runat="server">`. Vom umple *repeater*-ul cu un *DataSet* obţinut din interogarea bazei de date.



Jucătorii unei echipe

În funcţia `private void Page_Load(object sender, System.EventArgs e)` din clasa `public class Players : System.Web.UI.Page`, vom avea:

```
DataSet ds = Bridge.GetPlayers(teamId);
players.DataSource = ds.Tables["MyTable"];
players.DataBind();
```

Astfel, vom conecta drept sursă la *repeater*-ul *players* o mulţime de date (`DataSet`) identificate prin `ds`. Funcţia din `Bridge`, denumită `GetPlayers (int teamId)`, are codul următor:

```
public static DataSet GetPlayers(int TeamId)
{
  SqlDataAdapter da = new SqlDataAdapter();
  SqlCommand command = new SqlCommand();
```

```
    //select
    string selectString ;
    selectString = "SELECT * FROM players  WHERE    ";
    selectString += "  pOwner =" + TeamId;
    command.CommandText = selectString;
    command.Connection = con;
    da.SelectCommand = command;
    DataSet ds = new DataSet();
    //umplem DataSet-ul
    da.Fill(ds, "MyTable");
    return ds;
}
```

În acest moment, putem afișa cu ajutorul *repeater*-ului informațiile încărcate din baza de date. Pentru fiecare articol (jucător, în cazul nostru), vom afișa numele, ID-ul, vârsta, forma fizică, experiența, naționalitatea, valoarea, salariul, numărul de goluri înscrise, numărul de cartonașe galbene primite, numărul de cartonașe roșii primite. *Template*-ul (șablonul) pentru fiecare articol se prezintă în modul următor:

```
<ItemTemplate>
<table class="tblNormal" width="90%" align="center">
<tr><td>
<table>
<tr>
<td class="rptHeader">
<a href="Player.aspx?id=
  <%#DataBinder.Eval(Container.DataItem, "pId")%>">
<%# DataBinder.Eval(Container.DataItem, "pFname") %>
<%# DataBinder.Eval(Container.DataItem, "pLname") %></a>
</td>
<td>(<%# DataBinder.Eval(Container.DataItem, "pId") %>)</td>
</tr>
<tr>
<td colspan="2">
<%# DataBinder.Eval(Container.DataItem, "pAge") %>  de ani,
forma
<%# Common.LevelForNumber(
  DataBinder.Eval(Container.DataItem, "pForm")) %>a
</td>
</tr>
<tr>
<td colspan="2">Este un tip  <%# Common.LevelForSeverity(
  DataBinder.Eval(Container.DataItem,"pSev"))%>
si are o experienta <%#Common.LevelForNumber(
  DataBinder.Eval(Container.DataItem, "pExp")) %>a
</td>
```

```
</tr>
<tr>
<td>
<table>
<tr>
<td class="rptItem">Nationalitate</td>
<td>
<%# Common.NatForId(DataBinder.Eval(
  Container.DataItem, "pNat"))%>
</td>
<td>
<asp:Image ID="flag" runat="server"
ImageUrl=<%#CreateImageUrl(DataBinder.Eval(
  Container.DataItem, "pNat"))%> ></asp:Image></td>
</tr>
<tr>
<td class="rptItem">Valoare</td>
<td><%# DataBinder.Eval(Container.DataItem, "pAval") %></td>
<td width="10%"> </td>
</tr>
<tr>
<td class="rptItem">Salar</td>
<td><%# DataBinder.Eval(Container.DataItem, "pWage") %></td>
<td width="10%"> </td>
</tr>
</table>
</td>
<td>
<table>
<tr>
<td class="rptItem">Goluri</td>
<td><%# DataBinder.Eval(Container.DataItem, "pGoals") %></td>
<td width ="2%"> </td>
</tr>
<tr>
<td class="rptItem">Cartonase galbene</td>
<td><%# DataBinder.Eval(Container.DataItem, "pYcard") %></td>
<td><asp:Image ID="Ycard"  Runat="server"
ImageUrl="Images/Cards/y.gif"></asp:Image>
</td>
</tr>
<tr>
<td class="rptItem">Cartonase rosii</td>
<td><%# DataBinder.Eval(Container.DataItem, "pRcard") %></td>
<td > <asp:Image ID="Rcard"  Runat="server" ImageUrl="Images/
Cards/r.gif"></asp:Image></td>
```

```
</tr>
</table>
</td>
</tr>
</table>
</td></tr></table>
</ItemTemplate>
```

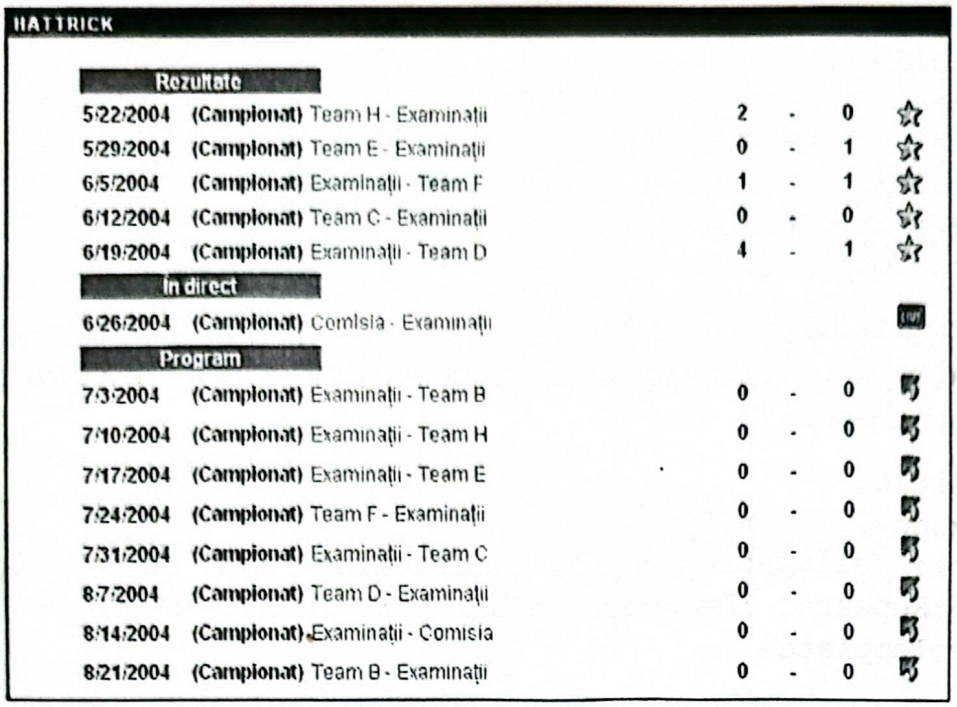
În pagina *Jucători* mai există, în partea dreaptă, un *fieldset* care conține legături rapide către toți jucătorii echipei. Acesta se realizează cu următoarea secvență de cod:

```
<fieldset>
<legend>Jucatorii tai</legend>
<table cellspacing="0px", cellpadding="0px" border="0px">
<asp:Repeater ID="ShortcutToPlayers" Runat="server">
<ItemTemplate>
<tr>
<td class="fieldPlayers">
<a  href="Player.aspx?id=<%#
DataBinder.Eval(Container.DataItem, "pId")%>">
<%# DataBinder.Eval(Container.DataItem, "pFname") %>
<%# DataBinder.Eval(Container.DataItem, "pLname") %>
</a>
</td>
</tr>
</ItemTemplate>
</asp:Repeater>
</table>
</fieldset>
```

Vizualizarea meciurilor unei echipe

În pagina *Meciuri* puteți vizualiza rezultatele meciurilor jucate, meciurile în direct și programul viitor. Pentru meciurile jucate, puteți accesa legături către „filmul" meciului și statisticile acestuia. Pentru meciurile desfășurate în direct, puteți accesa legătura către filmul meciului.

Încărcarea și afișarea meciurilor jucate, a meciurilor în direct și a meciurilor viitoare se realizează tot cu ajutorul unor funcții din modulul *Bridge* care returnează o mulțime de date (`DataSet`), pasat apoi ca sursă de date (`DataSource`) pentru un *repeater*.



Vizualizarea meciurilor

Pentru fiecare dintre cele trei tipuri de meciuri, se recurge la următoarea secvență de cod. Exemplul de mai jos este utilizat pentru afișarea meciurilor jucate. La fel se obțin și rezultatele pentru celelalte două tipuri de meciuri: meciurile *live* și meciurile din programul viitor.

```
<table align="center">
<asp:Repeater ID="matchesP" Runat="server">
<HeaderTemplate>
<tr>
<td colspan="7">
<asp:Label Runat="server" CssClass="labelStyle1"
            Width="150" ID="Label1">Rezultate</asp:Label>
</td>
</tr>
</HeaderTemplate>
<ItemTemplate>
<tr>
<td class="rptHeader" width="10%">
<%#GetDate(DataBinder.Eval(Container.DataItem, "mDate")) %>
</td>
<td class="rptHeader" width="10%"><%#
Bridge.GetTypeOfMatch(DataBinder.Eval(Container.DataItem,
  "mType")) %>
</td>
<td width="40%" >
<a href="Match.aspx?id=<%#
DataBinder.Eval(Container.DataItem, "mId") %>">
```

```
<%# Bridge.GetNameForTeam(DataBinder.Eval(
    Container.DataItem, "mHome")) %> - <%#
    Bridge.GetNameForTeam(DataBinder.Eval(
    Container.DataItem, "mAway")) %>
</a>
</td>
<td class="rptHeader" width="5%">
<%# DataBinder.Eval(Container.DataItem, "mHgoals") %>
</td>
<td class="rptHeader" width="5%"> - </td>
<td class="rptHeader" width="5%">
<%# DataBinder.Eval(Container.DataItem, "mAgoals") %>
</td>
<td class="rptHeader" width="5%">
<a href="Statistic.aspx?id=<%#
DataBinder.Eval(Container.DataItem, "mId") %>">
<img src="Images/stat.png" border="0">
</a>
</td>
</tr>
</ItemTemplate>
</asp:Repeater>
</table>
```

Atunci când dorim să vizualizăm filmul unui meci disputat, vom accesa *link*-ul reprezentat de scor. Pagina în care ajungem este obținută prin încărcarea informațiilor din baza de date pentru meciul cu ID-ul câmpului `mMovie`. În acest câmp avem stocat un fișier XML. Pentru a afișa filmul meciului, procesăm documentul XML. Despre modalitatea de prelucrare a datelor XML corespunzătoare filmului meciului, vom discuta puțin mai târziu.

Pentru a vizualiza statistica unui meci disputat, vom accesa steluța din dreptul meciului, pagina rezultată fiind obținută prin încărcarea din baza de date a valorilor câmpului `mStatistic` corespunzător meciului cu ID-ul respectiv. În pagina de statistici se afișează stelele pe care le-au obținut jucătorii în urma prestației din timpul meciului. Mai sunt disponibile informații despre ocaziile celor două echipe, golurile înscrise, cartonașele galbene, cartonașele roșii și posesia balonului.

Documentul XML obținut din baza de date este analizat de metoda:

```
private void ParseStatistics(string xml)
{
star [] Hplayers =  new star[11];
XmlDocument doc = new XmlDocument();
doc.LoadXml(xml);
XmlNodeList xnl = doc.GetElementsByTagName("home"); //acasa
XmlNode xn = xnl.Item(0); //stelutele
```

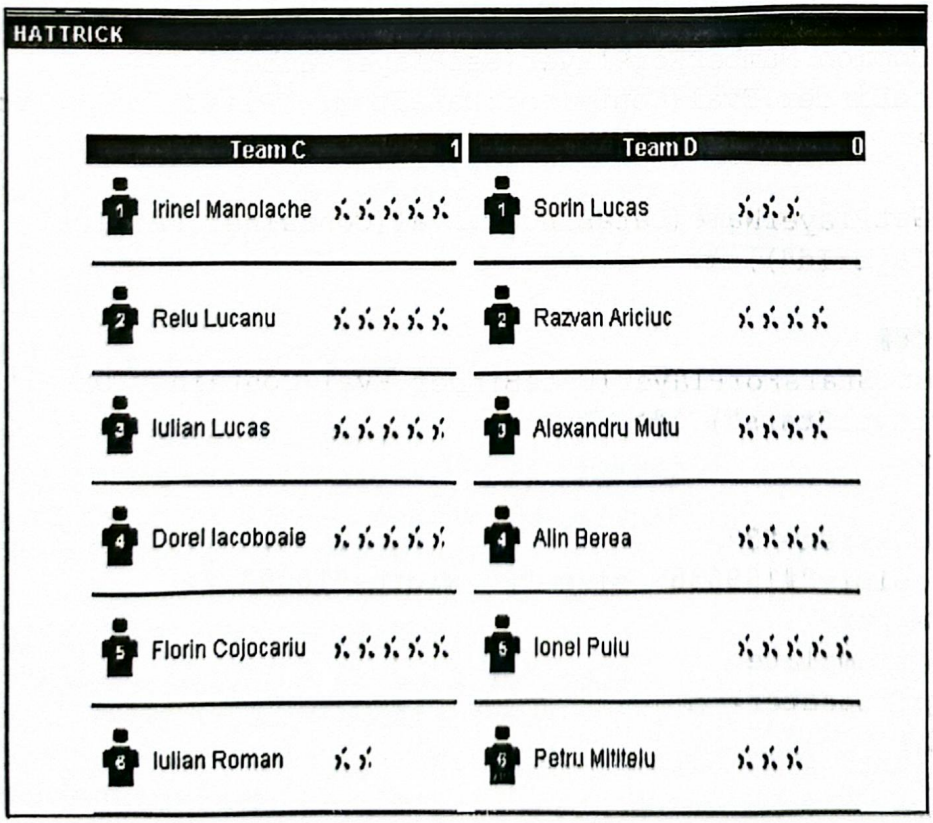
```
XmlNode stars = xn.ChildNodes[0];
XmlNodeList starList =stars.ChildNodes;
dsh = new DataSet();
DataTable t = new DataTable("Players");
t.Columns.Add(new DataColumn("PlayerId"));
t.Columns.Add(new DataColumn("PlayerStars"));
foreach(XmlNode st in starList)
{
  DataRow newRow = t.NewRow();
  newRow["PlayerId"] = Convert.ToInt32(
    st.Attributes.GetNamedItem("id").InnerText);
  newRow["PlayerStars"] = Convert.ToInt32(st.InnerText);
  t.Rows.Add(newRow);
}
dsh.Tables.Add(t);
home.DataSource = dsh;
home.DataBind();
XmlNode myNode = xn.ChildNodes[1]; //goluri
int goals = Convert.ToInt32(myNode.InnerText);
lblHGoalsValue.Text = myNode.InnerText;
ScoreHomeTeam.Text = myNode.InnerText;
myNode = xn.ChildNodes[2]; //ocazii
int misses = Convert.ToInt32(myNode.InnerText);
int occasions = goals + misses;
lblHOcassionValue.Text = occasions.ToString();
myNode = xn.ChildNodes[3]; //cartonase galbene
lblHycardsValue.Text = myNode.InnerText;
myNode = xn.ChildNodes[4]; //cartonase rosii
lblHrcardsValue.Text = myNode.InnerText;
myNode = xn.ChildNodes[5]; //posesii
lblHPossesionValue.Text = myNode.InnerText;
xnl = doc.GetElementsByTagName("away"); //auturi
xn = xnl.Item(0);
stars = xn.ChildNodes[0];
starList =stars.ChildNodes;
dsa = new DataSet();
t = new DataTable("Players");
t.Columns.Add(new DataColumn("PlayerId"));
t.Columns.Add(new DataColumn("PlayerStars"));
foreach(XmlNode st in starList)
{
  DataRow newRow = t.NewRow();
  newRow["PlayerId"] = Convert.ToInt32(
    st.Attributes.GetNamedItem("id").InnerText);
  newRow["PlayerStars"] = Convert.ToInt32(st.InnerText);
  t.Rows.Add(newRow);
}
```

```
dsa.Tables.Add(t);
away.DataSource = dsa;
away.DataBind();
myNode = xn.ChildNodes[1]; //goluri
goals = Convert.ToInt32(myNode.InnerText);
lblAGoalsValue.Text = myNode.InnerText;
ScoreAwayTeam.Text = myNode.InnerText;
myNode = xn.ChildNodes[2]; //ocazii
misses = Convert.ToInt32(myNode.InnerText);
occasions = goals + misses;
lblAOcassionValue.Text = occasions.ToString();
myNode = xn.ChildNodes[3]; //cartonase galbene
lblAycardsValue.Text = myNode.InnerText;
myNode = xn.ChildNodes[4]; //cartonase rosii
lblArcardsValue.Text = myNode.InnerText;
myNode = xn.ChildNodes[5]; //posesii
lblAPossesionValue.Text = myNode.InnerText;
}
```

Pentru a obține statistica, s-a folosit următoarea secvență de cod:

```
<table>
<tr><td>
<table>
<asp:repeater id="home" runat="server">
<ItemTemplate>
<tr><td>
<%# Common.NumberForPlayer(GetPlayerNumber(
  DataBinder.Eval(Container.DataItem, "PlayerId")), true) %>
</td>
<td>
<%# GetPlayerName(DataBinder.Eval(Container.DataItem,
  "PlayerId")) %>
</td>
<td><%#
Common.StarsForPlayer(DataBinder.Eval(Container.DataItem,
  "PlayerStars") )%></td>
</tr>
<tr>
<td colspan="3">
<hr color="#109636" size="2" width="100%" />
</td></tr>
</ItemTemplate>
</asp:repeater>
</table>
</td>
<td>
<table>
```

```
<asp:repeater id="away" runat="server">
<ItemTemplate>
<tr>
<td>
<%# Common.NumberForPlayer(GetPlayerNumber(
  DataBinder.Eval(Container.DataItem, "PlayerId")), false) %>
</td>
<td>
<%# GetPlayerName(DataBinder.Eval(Container.DataItem,
  "PlayerId")) %>
</td>
<td><%#
Common.StarsForPlayer(DataBinder.Eval(Container.DataItem,
  "PlayerStars") ) %></td></tr>
<tr>
<td colspan="3">
<hr color="#109636" size="2" width="100%" />
</td></tr>
</ItemTemplate>
</asp:repeater></table>
</td>
<td> </td>
</tr>
</table>
```

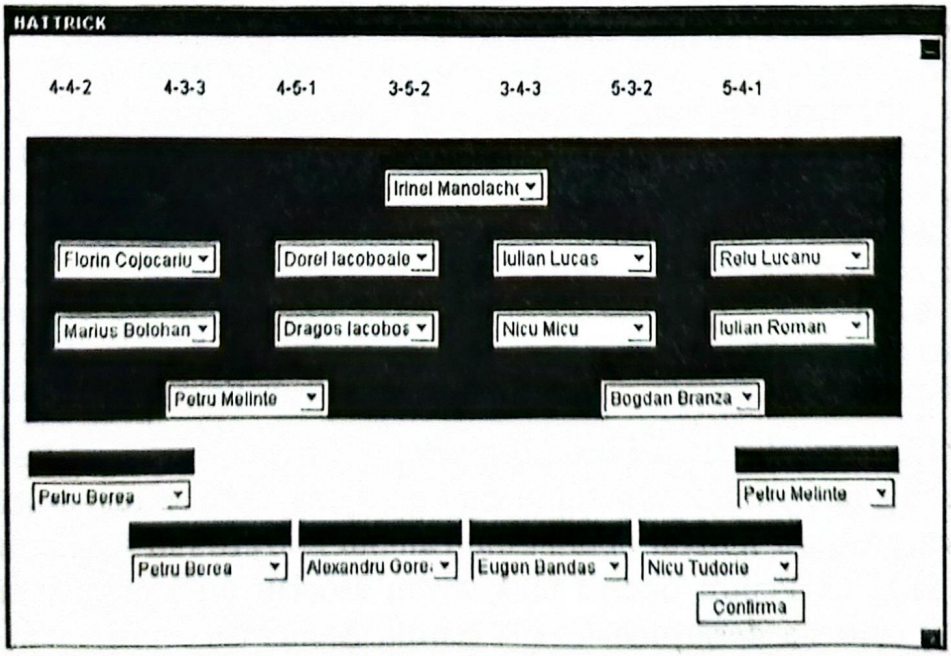


Statistica meciului

Au mai fost utilizate funcţiile de mai jos:

```
//furnizeaza numarul jucatorului
public string GetPlayerNumber( object id)
{
  DataSet aPlayer =
    Bridge.GetPlayer(Convert.ToInt32(id.ToString()));
  string result =
    aPlayer.Tables[0].Rows[0]["pNumber"].ToString();
  return result;
}
//furnizeaza numele jucatorului
public string GetPlayerName( object id)
{
  DataSet aPlayer =
    Bridge.GetPlayer(Convert.ToInt32(id.ToString()));
  string result = aPlayer.Tables[0].Rows[0]["pFname"] +
    " " + aPlayer.Tables[0].Rows[0]["pLname"];
  return result;
}
```

Pentru meciurile care nu s-au disputat încă, putem da ordine de joc. Dacă folosim *link*-ul din pagina *Meciuri* pentru un meci nedisputat încă şi care nici nu este în desfăşurare (un meci aflat sub titlul „Program"), vom ajunge într-o pagină în care putem alege o formulă de joc dintre următoarele: 4-4-2, 4-3-3, 4-5-1, 3-5-2, 3-4-3, 5-3-2, 5-4-1. Pentru fiecare dintre formule, utilizatorul poate selecta jucătorii pe care doreşte să-i alinieze în meciul respectiv.



Ordine pentru meci

Pagina este generată astfel: *combobox*-urile vor conține numele tuturor jucătorilor echipei, iar în fiecare dintre *combobox*-uri se va afla selectat numele unui jucător. Inițial, este o selecție aleatorie. După ce utilizatorul decide formula de echipă și jucătorii pe care îi va trimite în teren pentru meciul respectiv, selectându-i pe pozițiile dorite și apăsând butonul „Confirmă", pozițiile și jucătorii de pe acestea vor fi salvați în baza de date în câmpul corespunzător meciului respectiv.

În continuare, vom prezenta modul în care a fost implementată urmărirea filmului meciului în direct. Meciul efectiv este generat cu 15 minute înainte de ora oficială de începere a meciurilor dintr-o etapă. O dată ce meciul a fost generat, starea sa este stabilită ca fiind *live* (în direct). Astfel, meciul va apărea și în pagină cu starea *live*, iar atunci când va fi urmată legătura pentru a vedea filmul meciului, utilizatorul va putea urmări la început câte minute mai sunt până la startul meciului sau câte minute au trecut de la începutul meciului și fazele care s-au desfășurat. Minutele sunt date de funcția `GetMinuteOfTheMatch()`:

```
private int  GetMinuteOfTheMatch()
{
  int minute = 0;
  DateTime now = System.DateTime.Now;
  minute = (now.Hour - Config.SERVER_PLAY_MATCHES_START_HOUR)
    * 60 + now.Minute
    - Config.SERVER_PLAY_MATCHES_START_MINUTE;
  minute *= Config.SERVER_PLAY_MATCH_ACC;
  if (minute <= 45)
    return minute;
  else
    if (minute <= 60)
      return 45;
    else
      if (minute <= 105)
       retur· minute - 15;
      else
       return 90;
}
```

O dată ce ora oficială a startului meciului a trecut, încep să fie afișate fazele desfășurate pînă la minutul în care s-a ajuns. Cum se realizează acest lucru? Este procesat XML-ul obținut din câmpul `mMovie` din tabela *Matches* a bazei de date. Pentru fiecare fază, avem asociați un identificator (ID), minutul în care s-a desfășurat faza și scorul. Atunci când procesăm datele XML, ne vom opri din afișare în momentul în care minutul fazei pe care o prelucrăm depășește minutul de joc efectiv în care ne aflăm.

Fișierul XML poate avea următorul conținut:

```
<?xml version="1.0" encoding="iso-8859-2" ?>
<movie>
<h><j i="24"/><j i="25"/><j i="26"/><j i="27"/><j i="28"/>
<j i="29"/><j i="30"/><j i="31"/><j i="32"/><j i="33"/>
<j i="34"/></h>
<a>
<j i="0"/><j i="1"/><j i="2"/><j i="3"/><j i="4"/>
<j i="5"/><j i="6"/><j i="7"/><j i="8"/><j i="9"/><j i="10"/>
</a>
<p>
<f i="82" m="0" h="0" a="0">Scapat singur spre poarta
adversa, Aurel Ionescu a fost oprit prin fault de jucatorul
oaspetilor, Eugen Popa, acesta fiind eliminat pe loc.</f>
<f i="80" m="14" h="0" a="0">Cand Aurel Ionescu, absolut
neprovocat, a cosit un adversar chiar in fata tusierului,
arbitrul i-a aratat jucatorului gazdelor cartonasul rosu
fara ezitare.</f>
<f i="6" m="18" h="0" a="0">Jucatorul oaspetilor, Ciprian
Berea, a fost certat de antrenorul sau deoarece nu s-a
repliat la timp, gazdele fiind foarte aproape de a
inscrie.</f>
<f i="8" m="31" h="0" a="0">Jucatorul echipei Team B, Iulian
Calugaru, a refuzat sa paraseasca terenul dupa ce si-a ranit
tendonul lui Ahile in minutul 31, continuand jocul.</f>
<f i="40" m="41" h="0" a="0">O lovitura libera ce se anunta
periculoasa a fost tratata cu superficialitate de Ovidiu
Branza, oaspetii nereusind sa deschida scorul.</f>
<f i="15" m="42" h="0" a="1">Din apropierea coltului
terenului, Dumitru Bahmata a reusit totusi sa trimita o pasa
perfecta in centru, imposibil de ratat pentru Aurel Boian,
care a lovit pentru 1-0.</f>
<f i="72" m="44" h="0" a="1">Marian Bahmata de la gazde
a primit un cartonas galben in urma unor proteste la decizia
arbitrului de a semnala ofsaid.</f>
<f i="24" m="46" h="0" a="1">Golul egalizator pentru gazde
a parut o certitudine in minutul 46, in urma unei situatii de
unu la unu, dar Ionut Benea s-a remarcat, blocand sutul.</f>
<f i="80" m="54" h="0" a="1">Cand Iulian Calugaru, absolut
neprovocat, a cosit un adversar chiar in fata tusierului,
arbitrul i-a aratat jucatorului gazdelor cartonasul rosu fara
ezitare.</f>
<f i="15" m="65" h="0" a="2">Petru Haisan de la Keiser Sose
a inscris, aducand scorul la 2-0, dupa un sut puternic si
precis de la mare distanta, ce a lasat portarul advers fara
alta optiune decat aceea de a scoate balonul din plasa.</f>
```

```
<f i="45" m="71" h="0" a="2">Ion Moldovan a comis un fault
inutil chiar la marginea suprafetei de pedeapsa, dar lovitura
libera ce a urmat nu a avut efect pe tabela de marcaj.</f>
<f i="53" m="73" h="1" a="2">Dupa 73 de minute de la fluierul
de inceput al jocului, lui Team B i-a fost acordat un
penalti, pe care Aurel Bahmata l-a transformat in 1-2
fara probleme, reducand din diferenta.</f>
</p>
<best>28</best>
</movie>
```

Puteți observa că numele elementelor și ale atributelor XML au fost reduse la maximum, deoarece s-a încercat utilizarea a cât mai puțin spațiu în baza de date, confruntându-ne cu limitările impuse de SQL Server pentru lungimea unui câmp de tip `varchar` (4.000 de caractere maximum). De aceea, câmpul `mMovie` din baza de date este de tip text (acest tip de câmp poate conține date până la 2 MB). Pentru inserarea și citirea datelor din acest tip de câmp se folosesc funcții speciale (`WRITETEXT` și `READTEXT`).

În cadrul elementului rădăcină `movie`, avem patru elemente `h`, `a`, `p` și `best`. Identificatorul elementului `h` este o prescurtare pentru echipa care joacă acasă (*home*), iar elementele din interiorul *tag*-ului `h` (adică elementele de tip `j` care au ca atribut `i`) reprezintă, de fapt, ID-ul jucătorului. La fel este structurat și elementul `a`, care se referă la echipa care joacă în deplasare. Elementul de tip `p` (de la *play*) conține toate fazele din meci sub forma elementelor de tip `f` care au ca atribute: `i`, care reprezintă ID-ul fazei, `m`, care specifică minutul în care s-a desfășurat faza, iar `h` și `a` reprezintă scorul din minutul dat de atributul `m`. Valoarea lui `h` semnifică golurile înscrise de echipa gazdă pînă în acel moment, și `a` – golurile înscrise de echipa oaspete pînă în acel moment al jocului.

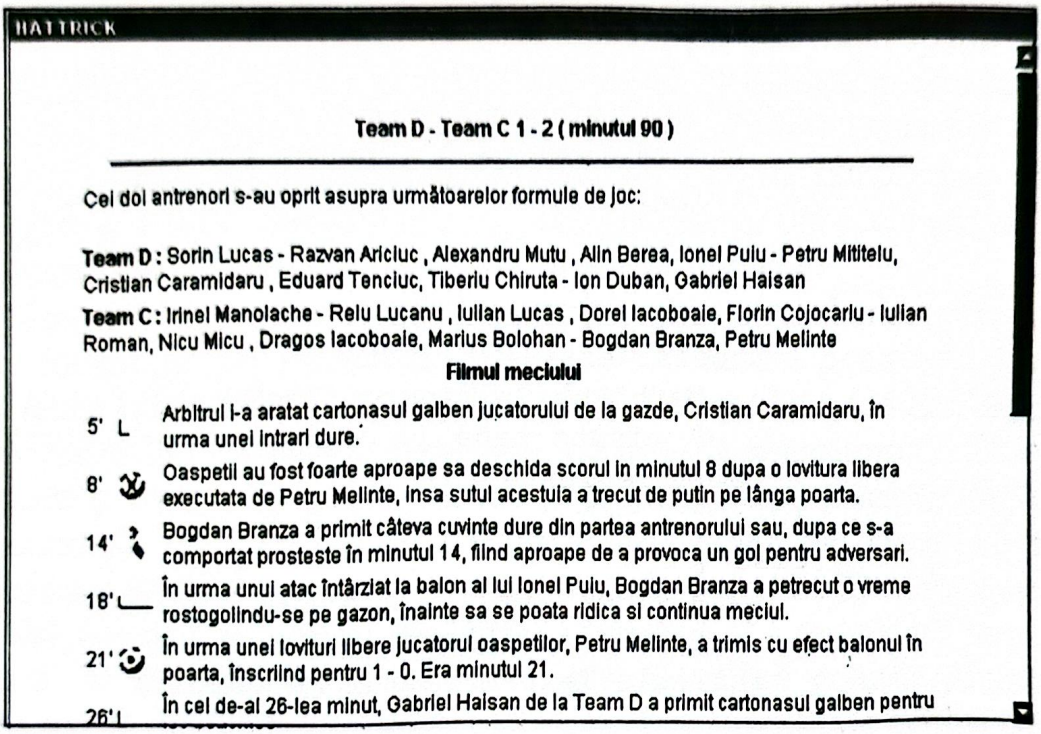
Codul din metoda `ParseMatch()` este următorul (folosim maniera de procesare XML din .NET, și anume *XMLReader*):

```
private void ParseMatch(string xml, DataSet ds)
{
      string IdAtt = "";
      string MinAtt = "";
      string Hgoals = "";
      string Agoals = "";
      int timeOfMatch;
      NameTable nt = new NameTable();
      XmlNamespaceManager nsmgr =
         new XmlNamespaceManager(nt);
      //Cream un context al procesarii XML
      XmlParserContext context =
         new XmlParserContext(null, nsmgr,
           null, XmlSpace.None);
```

```
//Cream cititorul XML
XmlTextReader reader =
    new XmlTextReader(xml,
    mlNodeType.Element, context);
bool phase = false;
Label1.Text = "<table>";
while (reader.Read()) // mai exista date XML
{
if (reader.IsStartElement("f")) { // un tag <f>
      IdAtt = reader.GetAttribute("i");
      MinAtt = reader.GetAttribute("m");
      timeOfMatch = Convert.ToInt32(MinAtt);
      if (timeOfMatch > GetMinuteOfTheMatch()) {
          Label4.Text += Hgoals + " - " + Agoals ;
          if(GetMinuteOfTheMatch() == 45) {
              Label4.Text += " ( pauza )";
              lblTitle.Text= "Filmul meciului";
          }
          else
          if(GetMinuteOfTheMatch() < 0) {
            Label4.Text += " ( " +
                Math.Abs(GetMinuteOfTheMatch()) +
                " minute pana la inceputul meciului)";
            lblTitle.Text= "";
          }
          else {
            Label4.Text += " ( minutul " +
                GetMinuteOfTheMatch() + " )";
            lblTitle.Text= "Filmul meciului";
          }
      return;
      }
      Hgoals = reader.GetAttribute("h");
      Agoals = reader.GetAttribute("a");
      Label1.Text += "<tr><td>" + MinAtt +
          "' </td>" + "<td>" +
          Common.GetPictureForPhase(IdAtt) + "</td>";
      phase = true;
}
else
if ( reader.HasValue ) {
  if ( phase ) {
      Label1.Text += "<td>" + reader.Value +
          "</td></tr>";
       phase = false;
  }
}//if
```

```
        }//while
        //afisam scorul si minutul partidei
        Label4.Text += ds.Tables[0].Rows[0]["mHgoals"]+
           " - "+ds.Tables[0].Rows[0]["mAgoals"];
        Label4.Text += " ( minutul " +
           GetMinuteOfTheMatch()+" )";
        reader.Close();
        Label1.Text += "</table>";
        lblTitle.Text= "Filmul meciului";
}
```

Rezultatul generat de această funcţie este următorul:



Filmul unui meci

Divizia A

Meniul include şi opţiunea *Divizia A*. Selectând această opţiune, putem vizualiza clasamentul Diviziei A şi golgheterii acesteia. Pentru parcurgerea clasamentului Divizei A, se foloseşte un *repeater*.

```
<form id="Form1" method="post" runat="server">
<table cellspacing="0">
<asp:Repeater ID="ranking" Runat="server">
<HeaderTemplate>
<thead>
<td class="labelStyle1">
<%# InitCounter(DataBinder.Eval(Container.DataItem, "gMatchesP"))%>
</td>
<td class="labelStyle1Left" width="40%">
```

```
<asp:Label ID="lblTeam" Runat="server"
CssClass="labelStyle1Left">Echipa</asp:Label>
</td>
<td class="labelStyle1" width="10%">
<asp:Label ID="lblPlayed" Runat="server">Meciuri
</asp:Label></td>
<td class="labelStyle1" width="10%">
<asp:Label ID="lblVict" Runat="server">Victorii</asp:Label></td>
<td class="labelStyle1" width="10%">
<asp:Label ID="lblDraws" Runat="server">Egaluri</asp:Label></td>
<td class="labelStyle1" width="10%">
<asp:Label ID="lblDef" Runat="server">Infrangeri
</asp:Label></td>
<td class="labelStyle1" width="10%">
<asp:Label ID="lblGoals" Runat="server">Goluri</asp:Label></td>
<td class="labelStyle1" width="10%">
<asp:Label ID="lblPoints" Runat="server">Puncte</asp:Label></td>
</thead>
<tr><td colspan="8"><br/></td></tr>
</HeaderTemplate>
<ItemTemplate>
<tr><td>
<%# Count(DataBinder.Eval(Container.DataItem, "gMatchesP"))%>.
</td>
<td width="40%" align="center">
<a href="Team.aspx?id=<%#DataBinder.Eval(Container.DataItem,
"gIdTeam")%>"> <%#
Bridge.GetNameForTeam(DataBinder.Eval(Container.DataItem,
"gIdTeam")) %></a></td>
<td  width="10%" align="center">
<%# DataBinder.Eval(Container.DataItem, "gMatchesP") %></td>
<td  width="10%" align="center">
<%# DataBinder.Eval(Container.DataItem, "gVict") %></td>
<td  width="10%" align="center">
<%# DataBinder.Eval(Container.DataItem, "gDraw") %></td>
<td  width="10%" align="center">
<%# DataBinder.Eval(Container.DataItem, "gDef") %></td>
<td  width="10%" align="center">
<%# DataBinder.Eval(Container.DataItem, "gGoalsS") %> -
<%# DataBinder.Eval(Container.DataItem, "gGoalsR") %></td>
<td  width="10%" align="center">
<%# DataBinder.Eval(Container.DataItem, "gPoints") %>
</td>
</tr>
<tr>
<td colspan="8">
<asp:Label ID="lblLine" Runat ="server">
```

```
<%# Line(DataBinder.Eval(Container.DataItem, "gPoints"))%>
</asp:Label></td></tr>
</ItemTemplate>
</asp:Repeater>
</table>
</form>
```

Setăm, ca `DataSource` pentru acest *repeater*, *DataSet*-ul returnat de metoda `GetGroup()` din *Bridge*.

```
public static DataSet GetGroup(int series, int group) {
      SqlDataAdapter da = new SqlDataAdapter();
      SqlCommand command = new SqlCommand();
      //select
      string selectString ;
      selectString= string.Format("SELECT * FROM {0}
        WHERE {1} = {2} AND {3}={4} ORDER BY gPoints DESC,
        gGoalsS-gGoalsR  DESC, gGoalsS DESC, gGoalsR DESC",
        "groups", "gDivision", series, "gGroup", group);
      command.CommandText = selectString;
      command.Connection = con;
      da.SelectCommand=command;
      DataSet ds = new DataSet();
      //umplem DataSet-ul
      da.Fill(ds, "MyTable");
      return ds;
}
```

Un posibil rezultat este următorul:

HATTRICK

	Echipa	Meciuri	Victorii	Egaluri	Înfrângeri	Goluri	Puncte
1.	Comisia	7	7	0	0	21-1	21
2.	Team H	7	6	1	0	11-0	19
3.	Team B	7	3	1	3	7-5	10
4.	Examinații	7	2	3	2	7-8	9
5.	Team F	7	2	2	3	8-9	8
6.	Team D	7	2	1	4	10-12	7
7.	Team C	7	1	1	5	6-21	4
8.	Team E	7	0	1	6	0-14	1

Clasamentul Diviziei A

Pentru vizualizarea clasamentului golgheterilor Divizei A, se folosește, de asemenea, un *repeater*, după cum puteți remarca în codul de mai jos:

```
<form id="Form1" method="post" runat="server">
<table width="100%">
<tr>
```

```
<td>
<table width="85%" align="left">
<asp:Repeater id="players" runat="server">
<HeaderTemplate>
<thead>
<td class="labelStyle1" width="5%">
<%# InitCounter(DataBinder.Eval(Container.DataItem,
  "pFname"))%>
</td>
<td class="labelStyle1" width="30%" align="left">
<asp:Label ID="lblName" Runat="server">Jucator</asp:Label>
</td>
<td class="labelStyle1" width="20%" align="left">
<asp:Label ID="Label1" Runat="server">Echipa</asp:Label>
</td>
<td class="labelStyle1" width="10%" align="left">
<asp:Label ID="lblGoals" Runat="server">Goluri</asp:Label>
</td>
</thead>
</HeaderTemplate>
<ItemTemplate>
<tr>
<td width="5%"><%# Count(DataBinder.Eval(Container.DataItem,
  "pFname"))%>. </td>
<td width="30%">  <a href="Player.aspx?id=
  <%#DataBinder.Eval(Container.DataItem, "pId")%>">
<%#DataBinder.Eval(Container.DataItem, "pFname")%>
  <%#DataBinder.Eval(Container.DataItem, "pLname")%>
</a></td>
 <td width="20%" align="center">
<%#Bridge.GetNameForTeamWithOwner(DataBinder.Eval(
   Container.DataItem, "pOwner"))%>
</td>
<td width="10%" align="center">
  <%#DataBinder.Eval(Container.DataItem, "pGoals")%>
</td>
</tr>
</ItemTemplate>
</asp:Repeater>
</table>
</td>
<td>
  <img src="Images/scorer.gif"/>
</td>
</tr>
</table>
</form>
```

Stabilim, drept `DataSource` pentru acest *repeater*, `DataSet`-ul returnat de metoda `GetGoalScorers()` definită în modulul *Bridge*.

```
public static DataSet GetGoalScorers(int number)
{
        SqlDataAdapter da = new SqlDataAdapter();
        SqlCommand command = new SqlCommand();
        string selectString ;
        selectString = "SELECT TOP " + number +
            " * from players  ";
        selectString += "ORDER BY pGoals DESC";
        command.CommandText = selectString;
        command.Connection = con;
        da.SelectCommand = command;
        DataSet ds = new DataSet();
        da.Fill(ds, "MyTable");
        return ds;
}
```

Un posibil rezultat este următorul:

HATTRICK

	Jucător	Echipa	Goluri
1.	Dan Tudorie	Team D	9
2.	Florin Donisa	Team F	8
3.	Viorel Ionescu	Team H	8
4.	Gheorghe Grigoras	Comisia	7
5.	Andrei Croitoru	Examinaţii	7
6.	Cosmin Damian	Team H	7
7.	Cristian Masalagiu	Comisia	6
8.	Viorel Chirica	Team F	6
9.	Daniel Puiu	Examinaţii	6
10.	Bogdan Popescu	Team B	5

Clasamentul Golgheterilor Diviziei A

3.5 De la SQL la XML – instrumentul SQLXML

Instrumentul Microsoft SQLXML 3.0 oferă posibilitatea creării de servicii Web care expun clienţilor funcţionalitatea *SQL Server*-ului. Se pot trimite cereri SOAP (*Simple Object Access Protocol*) serverului pe care rulează SQLXML 3.0, pentru a executa proceduri stocate, funcţii definite de utilizator (*UDFs – User Defined Functions*) sau şabloane (*template*-uri). Serverul pe care rulează SQLXML execută ceea ce i-a solicitat clientul şi îi returnează răspunsul SOAP corespunzător.

Scopul urmărit este cel de a crea un serviciu Web care să expună via SOAP diferite funcţionalităţi (de exemplu, selectarea jucătorilor unei echipe de fotbal) posibililor săi clienţi, funcţionalităţi care vor fi îndeplinite prin apelarea unor proceduri stocate în cadrul serverului de baze de date MS SQL Server. Astfel, SQLXML poate fi considerat ca o punte de legătură între un client şi serverul de baze de date, transferul datelor stocate în tabele efectuându-se în format XML via mesaje SOAP. Rezultatele interogărilor SQL (realizate de procedurile stocate) vor fi convertite „din zbor" în date XML spre a fi procesate ulterior de către clienţi. Protocolul folosit este SOAP, peste un protocol de transfer de nivel mai scăzut, precum HTTP.

Înainte de a trimite cererea SOAP către server, trebuie respectaţi următorii paşi:

1. Crearea unui director virtual. În acest director virtual trebuie să definim un nume virtual de tip SOAP.
2. Configurarea numelui virtual de tip SOAP. Aceasta produce un document WSDL (*Web Service Description Language*), descriind serviciul, alături de operaţiile (proceduri stocate, funcţii definite de utilizator şi *template*-uri) pe care un client le poate solicita. Amintim faptul că limbajul WSDL este folosit la descrierea într-o manieră independentă de platformă a serviciilor Web.

După crearea acestui document WSDL, se poate scrie aplicaţia client care să trimită spre server cererea SOAP pentru operaţiile specifice (definite în WSDL).

În procesul de configurare este creat şi un fişier de configuraţie (cu extensia *.ssc*), stocat în acelaşi director care este asociat numelui virtual de tip SOAP.

Crearea directorului virtual şi a numelui virtual de tip SOAP pot fi realizate utilizând *IIS Virtual Directory Management*, pentru SQLXML 3.0 sau, programatic, folosind *IIS Virtual Directory Management*, pentru modelul de obiect SQL Server.

Fiecare nume virtual de tip SOAP are asociat un fişier WSDL. Fişierul WSDL defineşte un serviciu format dintr-un set de metode.

Când se creează un client în Visual Studio.NET, se pot apela proceduri stocate, UDF-uri (funcţii definite de utilizator – *User Defined Functions*) sau *template*-uri, exact ca şi cum am apela metode ale oricărui obiect din aplicaţie. Rezultatul pentru proceduri stocate sau UDF-uri poate fi returnat astfel:

- `XMLElement (System.Xml.XmlElement)` – tablou de obiecte de acest tip;
- `DataSet (System.Data.DataSet)` – tablou de obiecte de acest tip;
- un singur obiect `DataSet (System.Data.DataSet)`.

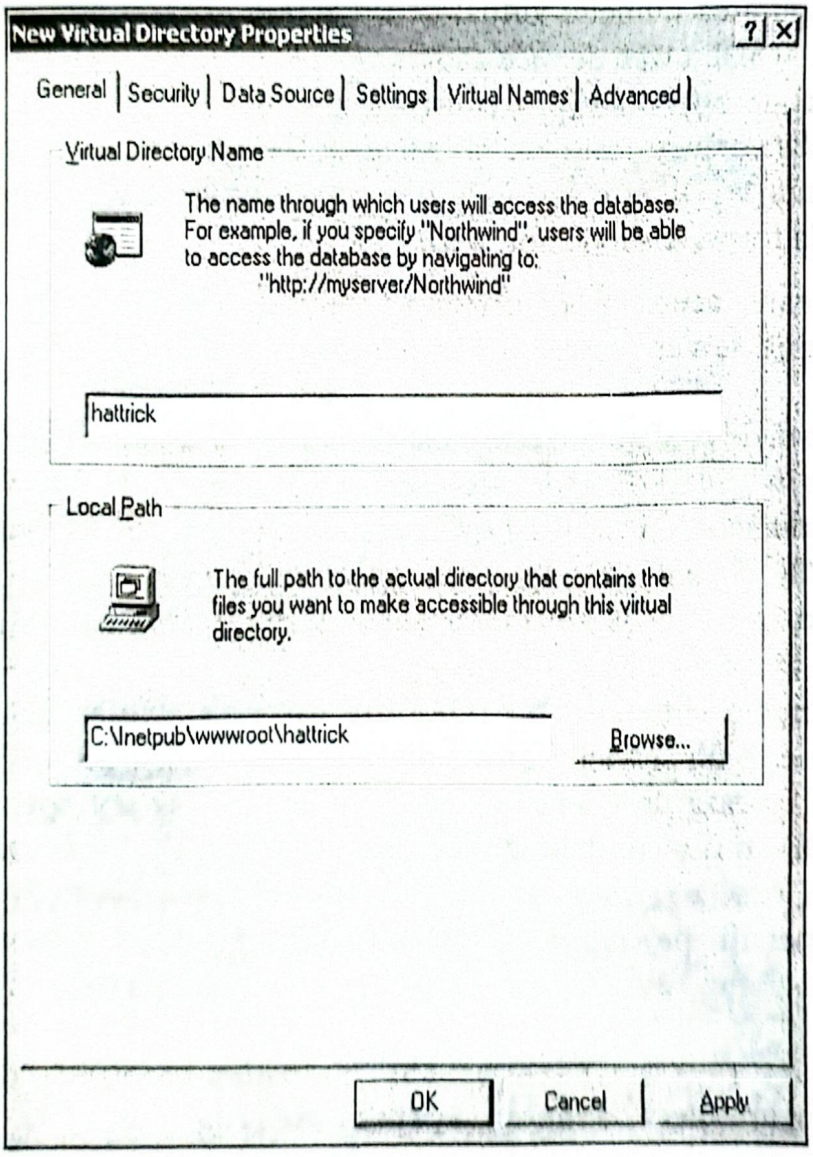
Pentru UDF-urile care întorc o singură valoare (de tip simplu), se returnează doar aceasta.

În cazul *template*-urilor, rezultatele sunt întotdeauna returnate într-un tablou cu elemente de tip `XMLElement`.

Maniera de returnare a rezultatelor depinde de modul de asociere a rezultatului, în procesul de configurare a numelui virtual de tip SOAP. Acest proces permite dezvoltatorului să specifice după dorință cum trebuie să fie returnat rezultatul.

Dacă Microsoft SQL Server returnează erori, acestea sunt tratate în funcție de modul de configurare a metodei de tratare (mod stabilit în cadrul configurării numelui virtual de tip SOAP). Pentru UDF-uri, erorile sunt întotdeauna returnate ca *SOAP faults*.

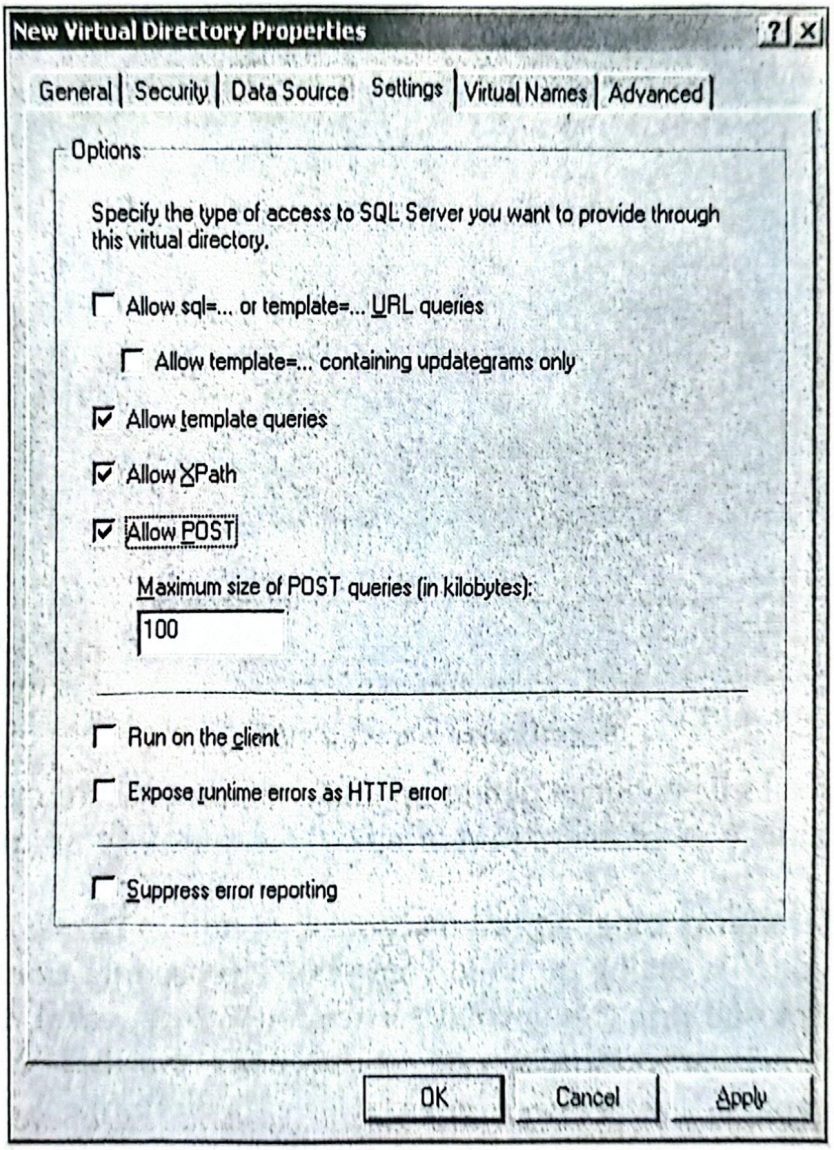
Vom încerca acum să descriem pașii urmați pentru a crea directorul virtual, numele virtual și fișierul WSDL folosit în aplicația noastră. Acestea au fost create cu ajutorul instrumentului *IIS Virtual Directory Management* pentru SQLXML 3.0. Se alege opțiunea *New Virtual Directory* și, în fereastra care apare, vom



Stabilirea directorului virtual

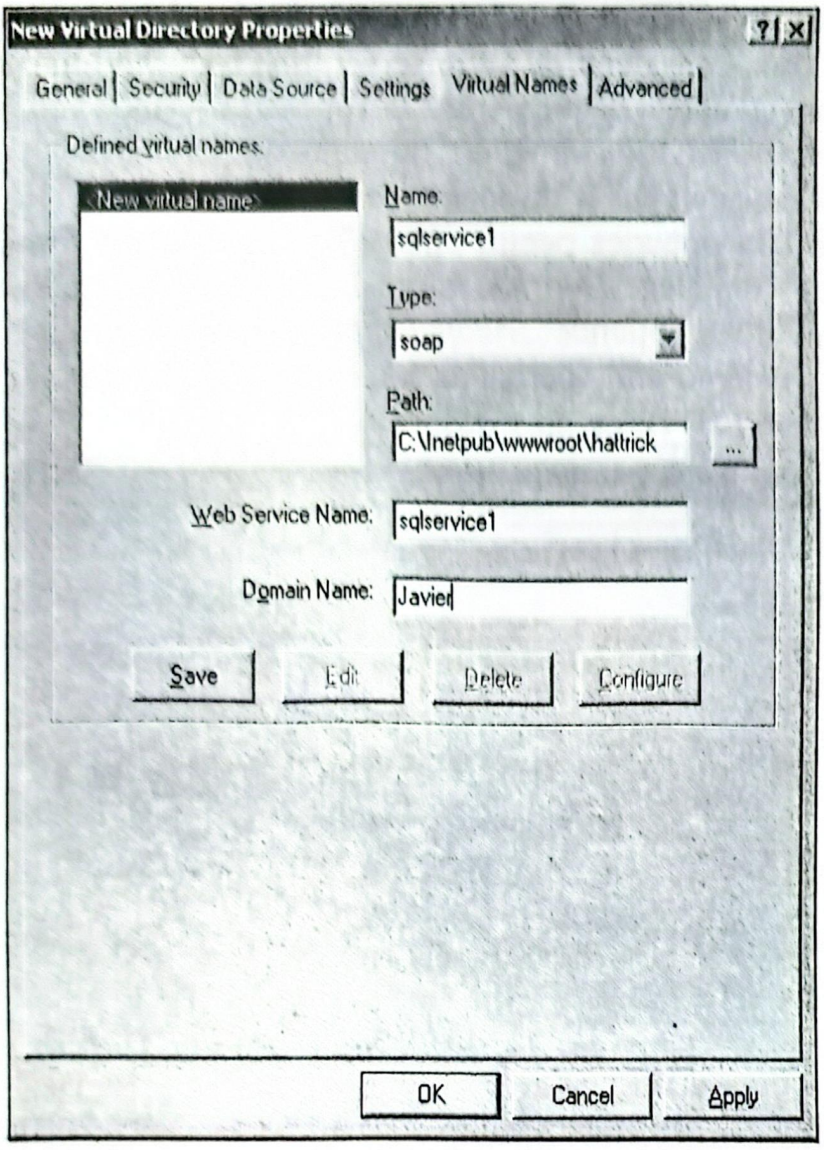
completa datele în *tab*-ul *General*: numele directorului virtual şi directorul fizic de pe disc, considerat de către serverul Web ca fiind director virtual.

Apoi, vom completa datele pentru *tab*-ul *Security*, unde vom selecta „*Use Windows Integrated Authentication*" (modulul de autentificare a utilizatorilor va folosi mecanismul pus la dispoziţie de sistemul Windows). În *tab*-ul *Data Source* vom lăsa opţiunea pentru SQL Server pe valoarea *local*, iar pentru baza de date vom alege *Hattrick*, creată în cadrul MS SQL Server. În *tab*-ul *Settings* vom alege opţiunile: „*Allow template queries*" (permitem cereri de şabloane), „*Allow XPath*" (permitem interogări XML XPath) şi „*Allow Post*" (permitem cereri HTTP de tip POST).



Stabilirea diverselor opţiuni

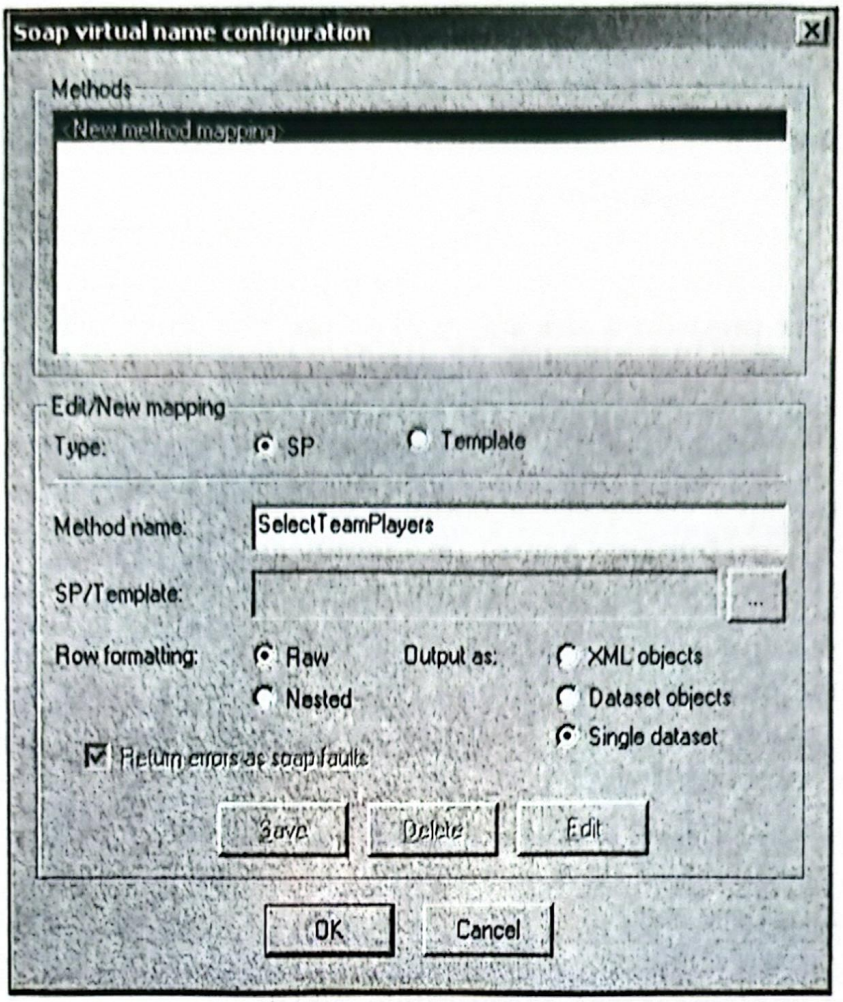
În *tab*-ul *Virtual Names*, vom defini numele virtual pe care îl vom folosi în continuare. Am introdus, pentru exemplul nostru, numele virtual „*sqlservice1*", tipul ales pentru acest nume fiind SOAP. Apoi, alegem calea directorului virtual asociat cu sursele de date (vezi mai sus).



Specificarea numelui virtual

Vom lăsa valorile implicite pentru opţiunile referitoare la numele serviciul Web şi la numele de domeniu. Vom apăsa butonul *Save* şi apoi butonul *Configure*.

În noua fereastră care apare, vom acţiona butonul *SP/Template* şi ne va fi deschisă o cutie de dialog de unde vom selecta procedura stocată pe care dorim să o folosim prin intermediul serviciului Web. În cazul nostru, vom selecta mai întâi procedura stocată `VerifyUser` (verifică accesul unui utilizator) şi apoi `SelectTeamPlayers` (selectează jucătorii echipei). Pentru aceste proceduri stocate, vom alege ca rezultatul final să fie de tip *Single dataset* şi vom salva fiecare procedură stocată, menită a fi asociată funcţionalităţilor expuse de serviciul Web.



Selectarea procedurilor stocate pentru a fi invocate via SOAP

Procedurile stocate selectate mai sus sunt următoarele:

```
CREATE PROCEDURE SelectTeamPlayers (@Id int ) AS
    SELECT * from players where pOwner = @Id
GO

CREATE PROCEDURE VerifyUser (@name nvarchar(10), @pass
nvarchar(10) ) AS
if not exists ( select * from users where uUser = @name
                and uPass = @pass)
    return  -1
else
    select uId from users where uUser = @name
    and uPass = @pass
    return
GO
```

În momentul acesta, apăsând butonul *OK* şi apoi *Apply*, vom putea vedea rezultatele configurării realizate mergând în directorul asociat ca director virtual, unde vom găsi documentul WSDL generat şi un fişier de configurare cu extensia *.ssc*.

3.6 Clientul SQLXML – descrierea funcționalităților

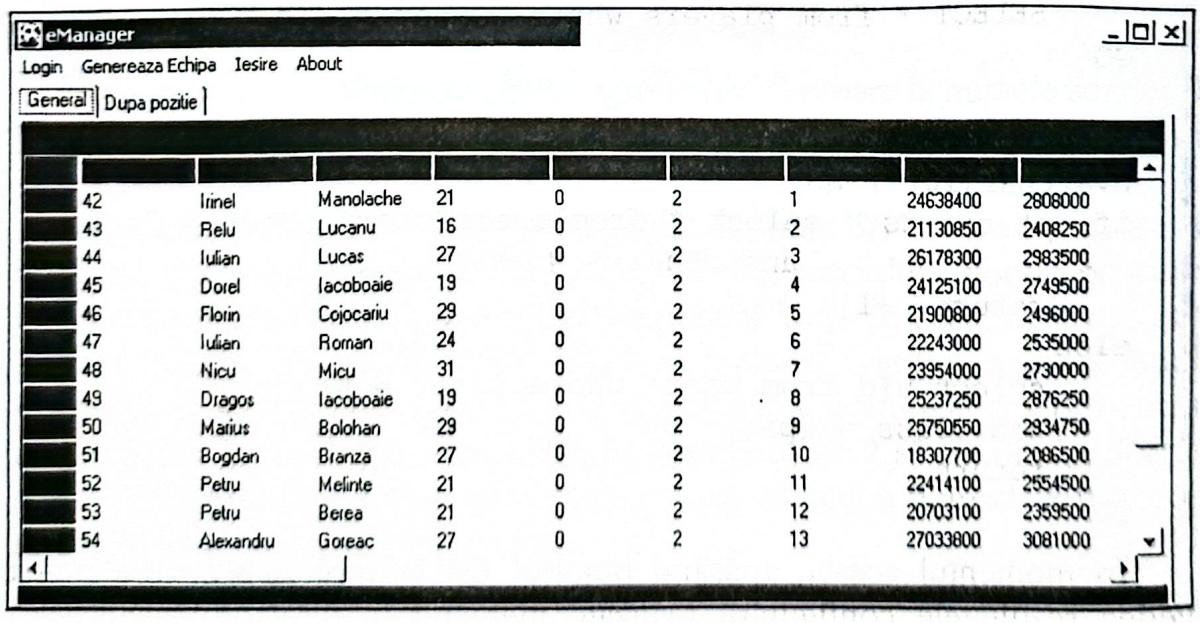
Acest client este folosit mai mult pentru a furniza informații privitoare la calitățile jucătorilor și a ajuta utilizatorul să-și formeze o idee despre cum ar trebui să-și așeze echipa în teren.

Pentru a folosi facilitățile acestui client, utilizatorul trebuie să furnizeze un nume de utilizator și o parolă. O dată introduse, se apelează un serviciu Web care – via procedura stocată `VerifyUser` descrisă mai sus – verifică existența și validitatea datelor introduse. Dacă datele sunt valide, se returnează codul utilizatorului, în caz contrar se returnează un cod de eroare.

După returnarea cu succes a codului utilizatorului, este apelat un nou serviciu Web, care întoarce pentru un identificator de utilizator toți jucătorii pe care acesta îi are ca jucători la clubul său virtual. În fapt, pe server este invocată procedura stocată `SelectTeamPlayers`.

După cum probabil cititorul și-a dat seama, toate serviciile Web sunt obținute cu ajutorul SQLXML 3.0, care ne ajută la expunerea – ca servicii Web – unor proceduri stocate pe serverul Microsoft SQL Server. Pentru a putea apela un astfel de serviciu Web dintr-o aplicație Windows, platforma .NET oferă posibilitatea de a instanția un intermediar (*proxy*), cu ajutorul căruia putem accesa metodele serviciului Web creat anterior. *Proxy*-ul a fost generat automat cu ajutorul unui instrument pus la dispoziție de platforma .NET și care implementează specificațiile WSDL.

Din moment ce am introdus un nume de utilizator existent și o parolă validă, vom obține informațiile despre jucători într-un control de tip *DataGrid*.



42	Irinel	Manolache	21	0	2	1	24638400	2808000
43	Relu	Lucanu	16	0	2	2	21130850	2408250
44	Iulian	Lucas	27	0	2	3	26178300	2983500
45	Dorel	Iacoboaie	19	0	2	4	24125100	2749500
46	Florin	Cojocariu	29	0	2	5	21900800	2496000
47	Iulian	Roman	24	0	2	6	22243000	2535000
48	Nicu	Micu	31	0	2	7	23954000	2730000
49	Dragos	Iacoboaie	19	0	2	8	25237250	2876250
50	Marius	Bolohan	29	0	2	9	25750550	2934750
51	Bogdan	Branza	27	0	2	10	18307700	2086500
52	Petru	Melinte	21	0	2	11	22414100	2554500
53	Petru	Berea	21	0	2	12	20703100	2359500
54	Alexandru	Goreac	27	0	2	13	27033800	3081000

Jucătorii unei echipe

Dând un dublu *click* pe unul dintre articolele *DataGrid*-ului, va apărea o fereastră care afișează structurat informațiile despre jucător și calitățile sale, plus un calcul al randamentului său pe toate posturile.

Acest client mai are o opțiune, cea de a crea o formulă de echipă, pe baza calităților pe care le au jucătorii. Utilizatorul va alege una dintre formulele de joc, iar clientul va genera o formulă de echipă, în care va așeza jucătorii pe pozițiile cele mai bune posibile în funcție de disponibilități. Implicit, va fi generată o echipă „ofensivă".

Vom explica în continuare maniera de generare a echipei. Se creează un tablou de date (de tip *DataTable*) în care așezăm pe coloane punctajul pe care l-ar avea un jucător pe posturile de portar, fundaș central, fundaș lateral, mijlocaș central, mijlocaș lateral și atacant. Dacă dorim să obținem o formulă ofensivă, vom sorta mai întâi jucătorii după punctajul de atacant și îi vom alege pe cei mai buni. Se setează câmpul `team = 1` pentru cei care au fost selectați în echipă, apoi se sortează după câmpul `mijlocas central` și sunt aleși jucătorii cu valoarea cea mai mare pentru acest câmp, dar care nu au fost deja selectați în echipă, iar apoi pentru jucătorii selectați în echipă se setează câmpul `team = 1`, continuându-se cu mijlocașii laterali, fundașii centrali, fundașii laterali și, la sfârșit, portarul.

În formularul generat se poate opta și pentru o formulă defensivă. Atunci când se alege această opțiune, se repetă algoritmul de mai sus, dar diferă ordinea de sortare a câmpurilor: se începe cu portarul și se continuă cu fundașul central, fundașul lateral, mijlocașul central, mijlocașul lateral și, în final, atacantul.

4. Concluzii

Prin capitolul de față se încheie un drum care a început exact acum un an, când am stabilit tema proiectului de licență. Stabilirea tehnologiilor folosite pentru implementare a fost un pas care ne-a dat prilejul să testăm și alte limbaje de programare (PHP). Considerăm că adoptarea platformei de lucru .NET Framework a fost cea mai bună alegere pentru îndeplinirea obiectivelor propuse. Integrarea coerentă a celor patru tipuri de aplicații (*Hattrick*, *eManager*, *HattrickManager* și *SqlXmlClient*) într-un singur proiect complex nu ar fi fost, probabil, posibilă utilizând alte limbaje sau paradigme de programare. În plus, prin acest proiect am putut acumula o bază de cunoștințe în tehnologiile .NET care cu siguranță ne va ajuta în viitor.

Determinarea și selectarea informațiilor care să corespundă jucătorilor, echipelor, întregii arhitecturi a cluburilor au reprezentat un alt stadiu important în dezvoltarea aplicației. Definirea tabelelor s-a realizat astfel încât proiectul să poată fi ulterior extins. Structura bazei de date actuale conține câmpuri care nu sunt utilizate în această versiune. Pozițiile jucătorilor corespunzătoare zonelor de teren, moralul echipei, locul de desfășurare a

meciului, forma jucătorilor sunt câteva dintre atributele care vor fi luate în calcul la următoarea versiune.

Dacă pentru utilizatori partea de server este mai puțin vizibilă, interfața grafică (pagina Web) reprezintă cea mai simplă modalitate de a asigura succesul aplicației. Folosirea noilor tehnologii Web, cum ar fi grafica vectorială prin SVG (vezi capitolul V) sau Flash-ul (vezi primele două capitole), poate fi considerată o soluție viabilă pentru prezentarea *live* a meciurilor.

În încheiere, putem afirma că ideea de a dezvolta o aplicație gen manager de fotbal *on-line* a fost o provocare care ne-a oferit șansa de a îmbina cunoștințele de informatică acumulate de-a lungul anilor cu pasiunea pentru fotbal.

5. Comentarii bibliografice

Pentru aprofundarea tehnologiilor utilizate, cititorii pot consulta următoarele lucrări apărute în ultimii ani:

- D.S. Platt, *Introducing Microsoft .NET*, Microsoft Press, 2003;
- T. Archer, *Inside C#*, Microsoft Press, 2002;
- S. Walther, *ASP.NET Unleashed*, Sams, 2001;
- J. Buyens, *Web Database Development Step by Step .NET Edition*, Microsoft Press, 2002.
- S. Short, *Building XML Web Services for the Microsoft .NET Platform*, Microsoft Press, 2002.

Alte materiale interesante sunt A. Turtschi *et al.*, *C# .NET Web Developer's Guide*, Syngress Publishing, 2002 și E. Brown, *Windows Forms Programming with C#*, Manning, 2002. De asemenea, se pot vizita siturile *http://www.gotdotnet.com/* și *http://msdn.microsoft.com/*.

O altă lucrare interesantă este *XML and ASP.NET*, avându-i ca autori pe K. Evans, A. Kamanna și J. Mueller, apărută la New Riders în 2002 și în care este prezentată importanța XML-ului în cadrul ASP.NET.

Pentru dezvoltarea și experimentarea funcționalității aplicațiilor ASP.NET, utilizatorii care nu posedă mediul Visual Studio.NET pot folosi un instrument pus la dispoziție gratuit – *ASP.NET WebMatrix* (vezi *http://www.asp.net/WebMatrix*).

În domeniul bazelor de date, de un real folos sunt cărțile C. Fleming, B. von Halle, *Handbook of Relational Database Design*, Addison-Wesley Professional, 1989, și C.J. Date, *An Introduction to Database Systems*, IBM Systems Programming Series, 1990. Mai recomandăm și M. Fotache, *SQL*, Polirom, 2001. Un sit Web care poate prezenta interes este *http://www.dbforums.com/*.

Specificațiile tehnice privitoare la SOAP și WSDL sunt disponibile la *http://www.w3.org/TR/*. Un alt sit util este *http://www.webservices.org/*.

Anexă: Structura tabelelor folosite de aplicaţia *Hattrick*

Tabela *teams* – conţine informaţii referitoare la echipe

Câmp	Tip de dată	Valori nule	Descriere
tId	*tinyint*		Identificator unic pentru o echipă (cheie primară)
tName	*nvarchar(50)*		Numele echipei
tSeries	*tinyint*		Seria din care face parte echipa
tGroup	*smallint*		Grupa în care joacă echipa
tRegion	*smallint*		Regiunea geografică de care aparţine echipa
tOwner	*smallint*	da	Proprietarul echipei (cheie secundară în tabela *users*)
tSuport	*int*		Numărul de suporteri
tConfid	*tinyint*		Moralul echipei
tDoctor	*tinyint*		Valoarea doctorului
tEconom	*tinyint*		Valoarea economistului
tPsyho	*tinyint*		Valoarea psihologului
tSname	*nvarchar(50)*		Numele stadionului
tSstatus	*tinyint*		Situaţia stadionului
tSseats	*tinyint*		Numărul de locuri de pe stadion
tEarena	*money*		Cheltuieli pentru întreţinerea arenei
tEwages	*money*		Salariile jucătorilor
tEcrowd	*money*		Banii obţinuţi din încasările de pe bilete
tEspons	*money*		Veniturile aduse de sponsori
tEexpt	*money*		Cheltuieli temporare
tEstaff	*money*		Cheltuielile cu personalul
tEyouth	*money*		Cheltuielile cu echipa de tineret
tEint	*money*		Dobânzi
tEtotal	*money*		Total disponibil
tStatus	*tinyint*		Situaţia echipei

Tabela *training* – conţine informaţii privitoare la antrenamente

Câmp	Tip de dată	Valori nule	Descriere
rId	*tinyint*		Identificator unic (cheie primară)
rTeam	*tinyint*		ID-ul echipei
rType	*tinyint*		Categoria valorii memorate (antrenament, antrenor secund etc.)
rNumber	*tinyint*		Numărul antrenorului, antrenamentului etc.
rValue	*tinyint*		Valoarea

Tabela *players* – conţine toţi jucătorii *Hattrick*

Câmp	Tip de dată	Valori nule	Descriere
pId	*smallint*		Cheie primară
pFname	*nvarchar(20)*		Prenume
pLname	*nvarchar(20)*		Nume
pAge	*tinyint*		Vârsta
pNat	*tinyint*		Naţionalitatea
pOwner	*smallint*	da	Proprietar (cheie secundară în tabela *teams*)
pNumber	*tinyint*	da	Numărul de joc
pAval	*int*		Valoarea estimativă
pWage	*int*		Salariul săptămânal
pGoals	*tinyint*		Numărul de goluri date
pYcard	*tinyint*		Numărul de cartonaşe galbene luate
pRcard	*tinyint*		Numărul de cartonaşe roşii luate
pStatus	*tinyint*		Informaţii de stare

Tabela *skills* – oferă informații privitoare la calitățile jucătorilor

Câmp	Tip de dată	Valori nule	Descriere
sId	*tinyint*		Identificator unic (cheie primară)
sPlayer	*smallint*		ID-ul jucătorului (cheie secundară în tabela *players*)
sSkill	*tinyint*		Identifică o calitate a jucătorului
sValue	*tinyint*		Valoarea jucătorului raportat la calitatea respectivă

Tabela *matches* – stochează meciurile și formulele de joc

Câmp	Tip de dată	Valori nule	Descriere
mId	*tinyint*		Identificator unic (cheie primară)
mDate	*datetime*		Data desfășurării partidei
mType	*tinyint*		Tipul meciului
mHome	*smallint*		ID-ul echipei gazdă
mAway	*smallint*		ID-ul echipei oaspete
mSpect	*int*	da	Numărul de spectatori prezenți la meci
mHform	*tinyint*		Formația de joc pentru echipa gazdă
mAform	*tinyint*		Formația de joc pentru echipa oaspete
mMovie	*text*	da	Descrierea desfășurării meciului
mStatistics	*nvarchar(1000)*	da	Statistica meciului
mHgoals	*tinyint*		Numărul de goluri înscrise de echipa gazdă
mAgoals	*tinyint*		Numărul de goluri înscrise de echipa oaspete
mStatus	*tinyint*		Situația meciului

Tabela *orders* – oferă date referitoare la poziţiile pe care sunt plasaţi jucătorii

Câmp	Tip de dată	Valori nule	Descriere
`oId`	*tinyint*		Identificator unic (cheie primară)
`oPlayer`	*smallint*		ID-ul jucătorului (cheie secundară în tabela *players*)
`oPosition`	*tinyint*		Poziţia pe care este plasat jucătorul

CAPITOLUL IX

De la *desktop* la dispozitive *wireless*

Adrian Lazariuc, adrian.lazariuc@spymac.com

Prezentul capitol expune modul de creare a unor aplicaţii pentru dispozitivele mobile, astfel încât acestea să comunice cu platforma *desktop* cât mai eficient. Pentru început, se prezintă principalele caracteristici ale platformei Palm OS, urmând apoi o incursiune în expunerea paşilor de realizare – folosind ASP.NET şi limbajul C – a unei aplicaţii, exemplificându-se întregul proces de dezvoltare de software dedicat dispozitivelor *handheld*. La final, se ilustrează posibilitatea utilizării practice a unui emulator Palm OS pentru testarea aplicaţiei implementate.

1. Tehnologia *handheld*

1.1 Ce reprezintă un dispozitiv *handheld*?

Handheld-urile sunt dispozitive extrem de portabile care conţin sisteme de management al informaţiilor şi pot realiza comunicaţii. Un candidat trebuie să treacă trei teste pentru a fi considerat un dispozitiv *handheld*:

- să opereze fără cabluri, cu excepţia unor cazuri temporare (reîncărcare, sincronizare cu calculatorul);
- să fie uşor de folosit direct în mână, fără a fi nevoie să fie aşezat pe masă;
- să permită adăugarea de aplicaţii sau conectarea la Internet – oferind facilităţi pentru *Wireless Application Protocol* (*WAP*), *i-mode* sau *e-mail*.

Handheld versus desktop

Platforma *desktop* este una staţionară, în timp ce platforma *handheld* este foarte portabilă. Între aceste două categorii se află calculatoarele care necesită o masă pentru a fi operate în mod eficient, precum *laptop*-urile şi computerele *palmtop*.

După cum se observă şi în tabelul următor, *handheld*-urile sunt mai mici, mai uşoare şi mai puţin „performante" decât *desktop*-urile. Oricum, ele oferă portabilitate şi acces foarte rapid la informaţii care necesită un timp de operare foarte mic.

Prezentăm, în continuare, un sumar comparativ al caracteristicilor celor două platforme principale:

	Handheld	*Desktop*
Motiv pentru a fi folosit	Introducere facilă a informaţiilor. Comunicare rapidă.	Abilităţi de prelucrare a unui volum foarte mare de informaţii. Acces la Internet (de exemplu, Web şi *e-mail*).
Cerinţe de formă şi utilizare	Mai puţin de 10 x 12". Poate fi folosit stând în picioare, fără o masă, în mod tipic putând să încapă într-un buzunar.	Necesită o masă; este folosit cel mai bine atunci când se stă jos. Uneori este portabil.
Mobilitate	Poate fi folosit uşor în timpul unei deplasări; bateria are o durată mare de viaţă.	Necesită un cablu de alimentare sau încărcări multiple.
Conectivitate	Lent şi greu de conectat, dar este în continuă dezvoltare.	Rapid şi uşor de conectat.
Modalităţi de introducere a datelor (Input)	*Stylus*, *keypad*, mini-tastatură, rotiţă	Tastatură şi mouse (există şi altele, dar mai puţin folosite pe scară largă).
Rezoluţia ecranului	240 x 320 pixeli sau mai puţin.	800 x 600 pixeli sau mai mult.

Sumar al caracteristicilor platformelor *bandeld* şi *desktop*

Dispozitivele *handheld* şi computerele de tip *desktop* sunt folosite pentru funcţii foarte diferite, deşi ambele oferă facilităţi pentru managementul informaţiei. Ambele tipuri pot să realizeze, de obicei, sincronizarea datelor. Aplicaţiile create pentru a fi utilizate pe un dispozitiv trebuie regândite pentru a fi exploatate pe alt tip de dispozitiv.

Computerele *desktop* sunt folosite pentru aplicaţii implicând procesarea unui volum mare de date, precum procesoare de text, analize financiare şi desenare asistată de calculator. Computerele *handheld*, din moment ce sunt mici şi uşoare, tind să fie utilizate ca managere personale de informaţii, vizualizatoare portabile pentru documente compresate şi ca dispozitive de introducere pe teren a datelor.

Utilizatorii de sisteme *desktop* rămân pe locurile lor ore întregi, câteodată. Utilizatorii de *handheld*-uri pot numai să arunce o privire asupra calendarului personal de evenimente pentru a vedea unde trebuie să meargă la o întâlnire

sau să găsească un număr de telefon, chiar înainte de a suna pe cineva, câteodată de pe același dispozitiv, chiar.

Handheld-urile pun la dispoziție acces rapid la informații, pe când computerele *desktop* necesită o interacțiune mai sofisticată cu utilizatorul.

1.2 Elemente caracteristice platformei Palm OS

Desigur, există mai multe arhitecturi (sisteme) de operare care fac posibilă operarea pe un dispozitiv *handheld*. Studiul nostru de caz a avut în vedere platforma Palm OS, una dintre cele mai cunoscute în domeniu.

Din punctul de vedere al programatorului, realizarea de aplicații pentru Palm OS se aseamănă, ca modalitate, cu dezvoltarea de aplicații pentru alte platforme, dar, uneori, acest proces se poate desfășura total diferit.

Două similarități importante sunt:

- aplicațiile sunt conduse de evenimente;
- se poate folosi orice limbaj, de la C standard până la *assembler*.

Diferențele tind să apară în cadrul unor facilități importante în ceea ce privește mărimea dispozitivului și scopul. Acestea includ modul în care Palm OS tratează cererile de memorie, aplicațiile și stocarea de date, precum și conectarea dispozitivului la calculatorul *desktop*.

1.3 Privire de ansamblu asupra sistemului Palm OS

Sistemul de operare Palm OS rulează pe baza unui nucleu preemptiv *multitasking*. Un *task* trebuie să ruleze interfața cu utilizatorul. Alte *task*-uri tratează diverse aspecte, precum monitorizarea introducerilor de date.

Interfața cu utilizatorul permite ca numai o aplicație să fie deschisă la un anumit moment. Astfel, atunci când o aplicație este deschisă, ea va ocupa întreg ecranul.

Vom descrie succint unele dintre caracteristicile principale în ceea ce privește managementul resurselor.

Memoria

Memoria este tratată într-un mod diferit față de maniera convențională. RAM-ul de pe un dispozitiv Palm OS este folosit pentru două scopuri:

- *Pentru alocare dinamică a memoriei* – aceasta este memoria de care aplicația are nevoie atunci când se execută. Ea include, de asemenea, coada de mesaje necesară aplicației. La resetare, această memorie este ștearsă, fiind analoagă cu memoria RAM a unui calculator tradițional.
- *Pentru stocare permanentă* – aceasta include aplicațiile *download*-ate, precum și datele pe care utilizatorii le vor vizualiza, crea și/sau edita.

Această parte nu este ştearsă la resetarea dispozitivului. Ea este analoagă memoriei reprezentate de un hard-disk.

Pentru ambele tipuri, alocarea este realizată progresiv. Partea de memorie pentru stocare permanentă stochează anumite baze de date (vezi mai jos). Spre deosebire de sistemele de operare *desktop* tradiţionale, datele şi codul nu sunt copiate de pe suportul permanent în memoria dinamică, în schimb sunt folosite direct din locul în care se află. De exemplu, atunci când codul unui program se execută, el este executat din zona de memorie unde e stocat, din moment ce partea permanentă este ea însăşi un fragment din RAM. În mod similar, datele pot fi citite şi afişate direct de la locaţiile în care sunt stocate.

Resursele

O aplicaţie Palm OS reprezintă o bază de date de resurse şi conţine mai multe tipuri diferite de resurse. O resursă este o înregistrare în baza de date şi are un tip şi un identificator (ID). Datele manipulate de aplicaţie sunt, de fapt, incluse în aceste resurse. În partea de cod (programabil) al aplicaţiei, resursele sunt definite în cadrul unui fişier cu extensia *.prc*.

Ca exemple de tipuri de date stocate în resurse, putem enumera: codul aplicaţiei, elementele interfeţei cu utilizatorul, şirurile de caractere, formularele sau pictogramele.

Elementele interfeţei cu utilizatorul sunt iniţializate pe baza conţinutului din aceste resurse. Deoarece iniţializarea nu este implementată în codul executabil al aplicaţiei, puteţi să modificaţi maniera de afişare a aplicaţiei dumneavoastră (de exemplu, folosirea altei limbi) fără a altera codul propriu-zis.

Un alt avantaj este acela că puteţi folosi editoare vizuale pentru a afişa şi a edita diferite zone ale interfeţei cu utilizatorul. Astfel de editoare vă permit să îmbunătăţiţi modul de prezentare a datelor, fără a recompila şi a retransfera de pe Internet aplicaţia.

Evenimentele

O aplicaţie Palm OS este condusă de evenimente (*event-driven*). La lansarea evenimentelor, precum apăsarea ecranului cu *stylus*-ul (stiloul Palm special) sau acţionarea unei taste, aplicaţia trebuie să răspundă la ele. Unele evenimente sunt tratate de aplicaţia dezvoltată de utilizator, altele sunt rezolvate de către sistemul de operare.

O dată ce aplicaţia începe să ruleze, ea intră într-o buclă de evenimente, primind în mod repetat şi apoi tratând câte un eveniment. Tratarea buclei continuă până când utilizatorul lansează o altă aplicaţie ori resetează dispozitivul.

Formulare şi controale

Palm OS are suport implicit pentru controale I/O şi pentru managementul formularelor. Formularele sunt similare ferestrelor unui sistem de operare *desktop*. Interfaţa cu utilizatorul fiind simplificată, numai un singur formular poate fi afişat la un moment dat. Palm OS pune la dispoziţie o interfaţă de programare (*API – Application Programming Interface*) bogată în ceea ce priveşte generarea şi prelucrarea formularelor, incluzând numeroase elemente de interacţiune cu utilizatorul.

Unele dintre cele mai importante componente sunt:

- *Checkboxes* (căsuţe cu bifare);
- *Radio buttons* (butoane radio);
- *Push buttons* (butoane cu apăsare);
- *Lists – one-column* (liste – cu o singură coloană);
- *Pickers – pop-up lists* (liste de tip *pop-up*);
- *Tables – multicolumn* (tabele – cu mai multe coloane);
- *Scrollbars* (bare de defilare);
- *Static text labels* (etichete statice);
- *Editable text fields* (câmpuri de text editabile);
- *Menus* (meniuri).

Comunicarea

Palm OS suportă o varietate de metode de comunicare. Protocoalele de comunicare curente sunt: comunicarea *serială* şi cea prin TCP/IP via interfaţa de programare *socket*.

Este furnizat un mecanism de interschimb de nivel înalt între obiecte, astfel încât să permită schimbul de informaţii dintre dispozitive Palm, precum şi cu alte dispozitive care folosesc un standard industrial pentru comunicarea obiectuală. Schimbul între obiecte se realizează via *IRDA*, deşi, în viitor, se preconizează să se furnizeze şi alte moduri de comunicare.

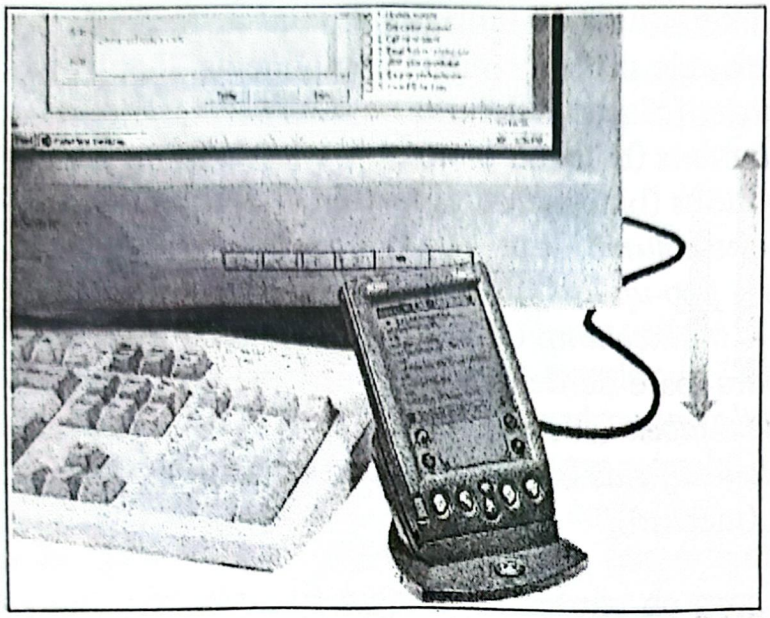
1.4 Prezentare succintă a protocoalelor de comunicaţie (*conduits*)

În afară de codul rulat pe dispozitivul *handheld*, a doua parte a unei aplicaţii Palm este reprezentată de conectivitatea cu *desktop*-ul. Deoarece dispozitivele Palm constituie o extensie a *desktop*-ului, este important ca informaţia să fie uşor de transmis. Protocoalele de comunicaţie (aşa-numitele conduite – *conduits*) reprezintă mecanismul pentru a realiza acest lucru.

Un protocol de comunicaţie (referit de către specialişti prin termenul *conduită*) reprezintă o porţiune de cod aflată pe *desktop* şi care este apelată în timpul unei sincronizări *HotSync* pentru a realiza managementul datelor

transmise de la și spre bazele de date de pe *handheld*. Conduitele înregistrează bazele de date pentru care sunt responsabile. Trebuie să precizăm faptul că fiecare bază de date trebuie să aibă numai o conduită responsabilă pentru ea.

Conduitele sunt create folosind instrumente de dezvoltare a conduitelor (CDK – *Conduit Development Kits*), disponibile pentru Windows (în limbajul C/C++) și Mac OS (în C/C++) sau pentru mediul Java.



Conectarea unui dispozitiv Palm OS la calculatorul obișnuit

Aplicațiile care nu au construită o conduită folosesc în schimb una furnizată de sistem. Această conduită specială este utilizată pentru realizarea copiilor de siguranță (*backup*) și face parte din procesul de sincronizare *HotSync*. Conduita de *backup* copie datele aplicației sau bazele de date de pe dispozitiv și le salvează într-un fișier pe calculatorul *desktop*.

Conduita de *backup* trebuie specificată atunci când se creează baza de date. În timpul unei sesiuni *HotSync*, conduita de *backup* este apelată pentru bazele de date care nu au o altă conduită și care au fost marcate ca având nevoie de un *backup*. În acest stadiu, ea copie fiecare înregistrare din baza de date și memorează antetul (*header*) acesteia într-un fișier. Acest fișier poate fi folosit pentru a restaura starea dispozitivului, dacă este nevoie.

Conduitele mai sofisticate realizează mai mult decât această copiere de informații. Ele pot să citească/scrie informații specifice pe sau de la *handheld*. De exemplu, o conduită pentru o aplicație *e-commerce* de vânzări poate să organizeze *download*-urile de noi liste de prețuri, suprascriind listele de prețuri care au expirat. Ea poate fi, de asemenea, responsabilă cu transferul unei baze de date cu ordine de vânzare.

Cele mai sofisticate conduite sunt cele care sincronizează înregistrările de pe *handheld* cu informația de pe *desktop*. Ca exemple putem da conduitele

care sincronizează informaţiile aplicaţiei *Date Book* (de pe Palm) cu cele ale programelor *Outlook* sau *Lotus Organizer* (de pe calculatorul *desktop*). Conduitele de sincronizare funcţionează, de obicei, prin atribuirea unui ID unic fiecărei înregistrări şi monitorizând modificările înregistrărilor.

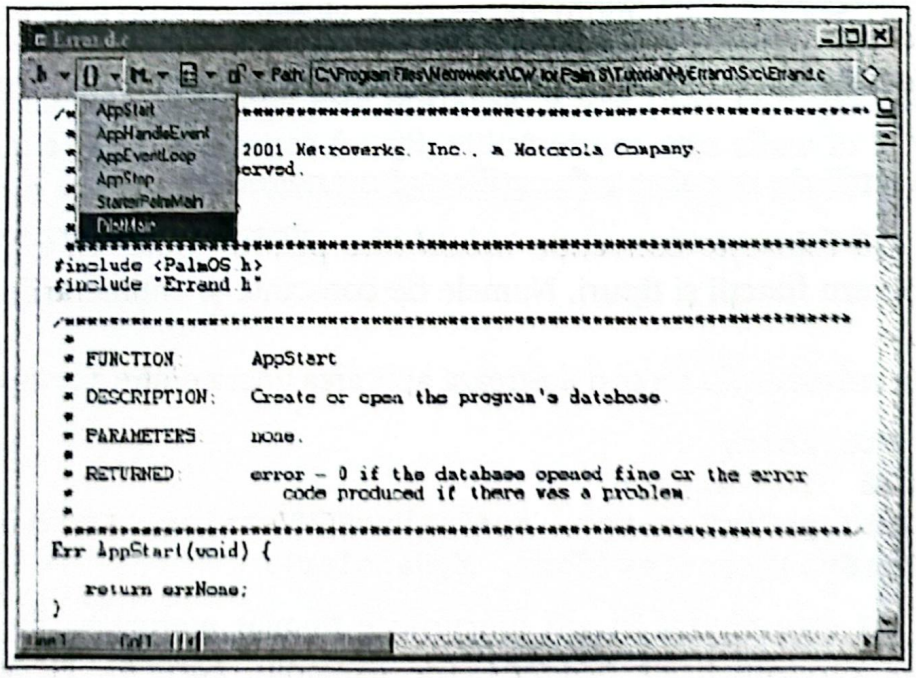
1.5 Medii de dezvoltare şi limbaje

Sunt disponibile o serie de instrumente care ajută programatorii să construiască, să testeze şi să depaneze aplicaţiile Palm OS. Unul dintre cele mai folosite programe este *CodeWarrior Integrated Development Environment* (*IDE*). Documentaţia pentru *CodeWarrior* IDE se găseşte inclusă în aplicaţie, iar pentru mai multe informaţii privitoare la alte medii de dezvoltare puteţi vizita adresa *http://www.palmos.com/dev/tools/*.

Caz similar multor aplicaţii tradiţionale, interfaţa cu utilizatorul se găseşte stocată într-unul sau mai multe fişiere de resurse. Puteţi folosi programul *Constructor* pentru Palm OS pentru a crea aceste resurse.

Pentru a realiza depanarea şi testarea aplicaţiei, există diverse unelte disponibile, dintre care menţionăm:

- *CodeWarrior Debugger* tratează depanarea la nivel de sursă (*source-level debugging*) şi poate fi folosit cu o aplicaţie rulând pe *handheld*-ul Palm OS sau în conjuncţie cu unul dintre instrumentele de dezvoltare;
- *Palm OS Emulator* testează aplicaţia pe un computer *desktop* înainte de a o transfera pe *handheld*;
- *Palm Debugger* este o unealtă de tip *assembly-level*. O puteţi folosi, de asemenea, pentru a introduce comenzi direct către *handheld*-ul Palm.



Captură de ecran din *CodeWarrior* la realizarea depanării aplicaţiei

2. Proiectarea aplicațiilor Palm OS

2.1 Convenții Palm OS

Există o varietate de convenții pentru scrierea de cod pentru Palm, care trebuie cunoscute: conversii de tip și modalități standard de numire pentru elementele folosite, de la funcții la manageri.

Tipuri de date

Principalele tipuri de date utilizate de Palm OS sunt:

- *UInt32* – întreg fără semn, pe 32 de biți;
- *Int32* – întreg cu semn, pe 32 de biți;
- *UInt16* – întreg fără semn, pe 16 biți;
- *Int16* – întreg cu semn, pe 16 biți;
- *UInt8* – întreg fără semn, pe 8 biți;
- *Int8* – întreg cu semn, pe 8 biți;
- *Boolean* – o valoare *true* sau *false*, pe 1 octet;
- *Char* – un caracter, pe 1 octet, care va putea fi utilizat numai pe sisteme cu codificare a caracterelor, pe 1 *byte* (ASCII);
- *WChar* – un caracter, pe 2 octeți, folosit pentru toate codificările (Unicode – incluzând japoneza, chineza etc.);
- *Err* – un întreg, pe 2 octeți, folosit pentru erori. Valoarea *errNone* indică faptul că nu există nici o eroare;
- *Coord* – un întreg cu semn, pe 2 octeți, folosit pentru a reprezenta coordonatele unei ferestre;
- *MemPtr* – specifică un pointer, pe 4 octeți, la o porțiune de memorie alocată;
- *MemHandle* – o referință, pe 4 octeți, la o porțiune de memorie realocabilă.

Convenții de numire a funcțiilor și managerilor

Palm OS folosește convenția *mixed-case* pentru nume, cu o majusculă inițială pentru funcții și tipuri. Numele de constante și enumerările încep cu minuscule.

Următorul exemplu de cod ilustrează aplicarea unora dintre aceste convenții:

```
FormPtr form;
UInt32  romVersion;
FtrGet(sysFtrCreator, sysFtrNumROMVersion, &romVersion);
FrmSetEventHandler(form, MyHandler);
```

Palm OS este divizat în arii funcționale numite *manageri*. Fiecare *manager* are propriul fișier *header* C (de exemplu, *Form.h*). Fiecare rutină dintr-un *manager* începe, de obicei, cu o abreviere pentru managerul

respectiv (de exemplu, *EvtGetEvent* este o parte a *Event Manager*, declarat în *Event.h*; *FrmGotoForm* provine din *Form Manager*, definit în *Form.h*). Tabelul următor conţine exemple de abrevieri:

Abreviere	Manager
Alm	*Alarm Manager* (Managerul de alerte)
Dm	*Data Manager* (Managerul de date)
Evt	*Event Manager* (Managerul de evenimente)
Ftr	*Feature Manager* (Managerul de facilităţi)
Mem	*Memory Manager* (Managerul de memorie)
Snd	*Sound Manager* (Managerul de sunet)
Str	*String Manager* (Managerul de caractere)
Sys	*System Manager* (Managerul de sistem)
Txt	*Text Manager* (Managerul de text)

Abrevierile standard pentru manageri

Convenţii pentru bibliotecile C

Probabil nu veţi folosi rutine ale bibliotecilor C standard, precum `strlen()` sau `memcpy()`. Din moment ce aceste funcţii sunt disponibile în biblioteci, a le utiliza în cod conduce la mărirea spaţiului de memorie ocupat de aplicaţie.

Din moment ce memoria este restrânsă, a fost pusă la dispoziţie o altă cale pentru a folosi asemenea funcţii, utilizându-se cele aflate în ROM, care ocupă puţin spaţiu şi sunt rapide. Următorul tabel prezintă rutinele Palm OS echivalente celor C standard:

Funcția C standard	Rutina Palm OS	Informații suplimentare
strlen	StrLen	
strcpy	StrCopy	
strncpy	StrNCopy	
strcat	StrCat	
strncat	StrNCat	Ultimul parametru reprezintă lungimea totală a șirului (incluzând terminatorul nul).
strcmp	StrCompare	
strncmp	StrNCompare	
itoa	StrIToA	
strchr	StrChr	
strstr	StrStr	
sprintf	StrPrintF	Subset limitat al lui `sprintf`, de exemplu fără `%f`.
svprintf	StrVPrintF	Subset limitat al lui `svprintf`, de exemplu fără `%f`.
malloc	MemPtrNew	
free	MemPtrFree	
memmove	MemMove	
memset	MemSet	
memcmp	MemCmp	

Echivalențe între funcțiile C standard și cele Palm OS

2.2 Palm OS și aplicațiile

Când Palm OS dorește să comunice cu o aplicație care poate să nu ruleze la un moment dat, el apelează rutina `PilotMain()` a acesteia.

Rutina principală: `PilotMain()`

Acesta reprezintă principalul punct de intrare într-o aplicație Palm OS și există întotdeauna o funcție numită astfel (într-un mod similar existenței funcției `main()` în cazul unui program C convențional).

Primul parametru desemnează codul de lansare, care specifică de ce este apelată funcția. De fiecare dată când aplicația este deschisă normal, acest parametru va fi reprezentat de constanta `sysAppLaunchCmdNormalLaunch`. Al doilea și al treilea parametru sunt folosiți atunci când aplicația este deschisă în alte contexte.

Un exemplu este furnizat în continuare:

```
UInt32 PilotMain(UInt16 launchCode,
  MemPtr launchParameters, UInt16 launchFlags)
{
#pragma unused(launchParameters)
  Err error;

  switch (launchCode) {

  case sysAppLaunchCmdNormalLaunch:
    /* incearca sa lanseze aplicatia */
    error = RomVersionCompatible
      (kOurMinVersion, launchFlags);
    if (error != errNone)
      return error;
    error = AppStart();
    if (error != errNone) /* eroare de lansare */
      return error;

    FrmGotoForm(MainForm); /* afiseaza forma principala */
    AppEventLoop(  ); /* proceseaza evenimentele */
    AppStop(  );      /* opreste aplicatia */
    break;

  default:
    break;
  }

  return errNone;
}
```

În cazul în care primul parametru are valoarea `sysAppLaunch CmdNormalLaunch`, prima dată verificăm dacă este suportată versiunea dispozitivului pe care rulăm aplicația.

Pragma este o directivă a compilatorului C. În cazul nostru, `#pragma unused(launchParameters)` indică aplicației *CodeWarrior* că parametrul `launchParameters` nu este folosit în funcție. Utilizând această directivă, *CodeWarrior* nu va mai trimite avertismente programatorului asupra faptului că un parametru nu este utilizat nicăieri.

La început, apelăm rutina `AppStart()`, care realizează inițializările specifice aplicației. Apelul lui `FrmGotoForm()` specifică faptul că forma `MainForm` va fi inițial afișată. Ea va adăuga un eveniment `frmLoadEvent` în *Event Manager*.

Va rula apoi motorul aplicației noastre – `AppEventLoop()` – până când utilizatorul închide programul. În acel moment, tratăm terminarea aplicației în funcția `AppStop()`.

Rutina de start : `AppStart()`

Aici tratăm deschiderea și inițializarea standard ale aplicației. Într-o aplicație tipică Palm OS, funcția va include deschiderea bazelor de date și citirea informațiilor de la utilizator.

```
static Err AppStart(void)
{
    return 0;
}
```

Specificarea faptului că această funcție este statică garantează că nu va fi apelată din afara acestui fișier.

Rutina de terminare a aplicației: `AppStop()`

În mod normal, în `AppStop()` tratăm toate operațiile de închidere, precum închiderea bazelor de date, salvarea informațiilor despre starea curentă etc.

Funcția `FrmCloseAllForms` va închide toate formele deschise.

```
static void AppStop(void)
{
  FrmCloseAllForms();
}
```

Coada principală de evenimente

În `PilotMain()`, după inițializare, se apelează `AppEventLoop`. Pe scurt, iată ce se întâmplă în această buclă de evenimente:

- procesăm evenimente în mod continuu, tratându-le atunci când sistemul permite acest lucru;
- parcurgem bucla, selectând un eveniment cu `EvtGetEvent()`, iar apoi îl expediem unuia dintre cele patru *handler*-uri de evenimente;
- dacă unul dintre acestea returnează *true*, înseamnă că a tratat evenimentul și nu trebuie să îl mai procesăm în continuare;
- `EvtGetEvent()` primește apoi următorul eveniment din coada de evenimente;
- bucla de evenimente continuă în această manieră, până când se întâlnește evenimentul `appStopEvent`, iar atunci se iese din buclă și se realizează terminarea aplicației în funcția `AppStop()`.

Iată un exemplu de funcție `AppEventLoop()` tipică:

```
void AppEventLoop(void)
{
   Err error;
   EventType event;

   do {
      /* preia un eveniment */
      EvtGetEvent(&event, evtWaitForever);

      /* daca nu l-a tratat, continua */
      if (! SysHandleEvent(&event))
         if (! MenuHandleEvent(0, &event, &error))
            if (! AppHandleEvent(&event))
               FrmDispatchEvent(&event);

   } while (event.eType != appStopEvent);
/* se continua atat timp cat evenimentul primit
   este diferit de appStopEvent */
}
```

2.3 Folosirea memoriei în aplicații

Introducere

După cum am menționat, memoria este o resursă extrem de limitată pe dispozitivele Palm OS. Din această cauză, aplicațiile Palm OS trebuie să fie scrise cu foarte mare atenție în ceea ce privește managementul memoriei.

RAM-ul este divizat în două arii: de stocare și dinamică. Partea de memorie folosită pentru stocare este organizată de *Database Manager* (managerul de baze de date), iar memoria dinamică – de *Memory Manager* (managerul de memorie).

Memoria folosită pentru stocare (*Storage Area*)	Memoria instalată pe dispozitivul Palm OS
Memoria dinamică (*Dynamic Area*)	Variabile globale ale sistemului (*System Globals*)
	Alocări dinamice rezervate sistemului (*System Dynamic Allocation*)
	Alocări dinamice rezervate aplicațiilor (*Application Dynamic Allocation*)
	Stiva alocată aplicațiilor (*Application Stack*)
	Variabile globale ale aplicațiilor (*Application Globals*)

Organizarea memoriei pe un dispozitiv Palm OS

Memoria dinamică este folosită pentru stocarea variabilelor globale ale sistemului Palm OS în contextul alocărilor dinamice de memorie, a variabilelor globale în cadrul aplicațiilor utilizatorului, a cozii de evenimente a aplicației. De asemenea, este utilizată pentru orice alocări dinamice de memorie solicitate de către programator.

Dynamic Heap

Memoria dinamică este numită *dynamic heap*. Din această zonă de memorie se pot aloca părți *nonrelocatable* (numite pointeri) sau *relocatable* (numite *handles*).

Pentru a consulta sau altera conținutul unui bloc de tip *relocatable*, acesta trebuie temporar blocat. Atunci când are loc o alocare de memorie, orice bloc *relocatable* neblocat poate fi realocat.

Memory API

Acest API este folosit pentru a lucra cu memoria. Funcția `MemHandleNew()` permite alocarea unui *handler* astfel:

```
VoidHand myHandle = MemHandleNew(chunkSize);
```

Se va returna NULL dacă alocarea de memorie nu s-a realizat cu succes.

Înainte de a citi sau a scrie dintr-un, respectiv într-un *handler*, trebuie să îl blocați. Acest lucru este realizat prin apelarea funcției `MemHandleLock()`, care returnează un pointer la datele blocate. În timp ce *handler*-ul este blocat, blocul realocabil nu poate fi mutat și puteți să realizați acțiuni precum citirea și scrierea datelor. În general, ar trebui să mențineți un *handler* blocat cât mai puțin timp posibil; *handler*-urile blocate tind să fragmenteze memoria rămasă liberă.

Codul pentru a bloca un *handler* de memorie poate fi următorul:

```
void *myPointer = MemHandleLock(myHandle);
/* realizam o actiune cu pointerul */
```

`MemHandleUnlock()` decrementează contorul pentru blocări. De notat faptul că nu este permis ca porțiunea de memorie să se mute decât atunci când valoarea contorului asociat acesteia este 0. Dacă încercați să blocați un *handler* care a mai fost blocat de alte 15 ori, veți primi un mesaj de eroare „*chunk overlocked*". În mod similar, deblocarea unui *handler* care este deja deblocat (deci care are contorul 0) generează un mesaj de eroare „*chunk underlocked*".

Pentru a aloca un pointer se poate recurge la funcția `MemPtrNew()`:

```
struct s *myS = MemPtrNew(sizeof(struct s));
```

Pentru a elibera o parte de memorie, folosiți `MemPtrFree()` sau `MemHandleFree()`:

```
MemPtrFree(myPointer);
MemHandleFree(myHandle);
```

Atunci când o porțiune de memorie este alocată, aceasta este marcată cu un identificator (ID) al proprietarului său. În cazul în care o aplicație este închisă, managerul de memorie anulează alocările tuturor porțiunilor de memorie având ID-ul proprietarului respectiv. Alte zone (de exemplu, cele alocate de sistem cu diferite marcaje) rămân alocate.

Nu trebuie să vă bazați însă pe această „curățare" a memoriei. În schimb, trebuie să scrieți cod în aplicație pentru a elibera în mod explicit toate porțiunile de memorie.

Această manieră de eliberare a memoriei face mai ușoară viața utilizatorilor de Palm. Există unele aplicații scrise necorespunzător care pot crea probleme cu privire la memoria alocată. Un dispozitiv Palm nu este asemeni unui computer *desktop* care poate fi *reboot*-at destul de des. Din contră, un dispozitiv Palm trebuie să ruleze luni de zile fără a fi resetat.

Suplimentar, există alte două rutine care lucrează cu memoria – `MemSet()` și `MemMove()`:

```
MemSet(void *p, ULong numBytes, Byte value);
MemMove(void *from, void *to, ULong numBytes);
```

Funcția `MemSet()` stabilește o anumită dimensiune pentru o zonă de memorie, iar `MemMove()` copie numărul de octeți specificat dintr-o anumită zonă în alta.

2.4 Crearea interfeței cu utilizatorul

Introducere

Palm OS pune la dispoziție o serie de resurse care reprezintă elemente ale interfeței cu utilizatorul. Partea vizibilă a unei aplicații Palm OS este locul în care au loc interacțiunile; astfel, este important să se știe ce controale există și modul în care acestea funcționează.

Fiecare element din cadrul interfeței cu utilizatorul este o resursă pe care o construiți și apoi o compilați în aplicația creată. În acest subcapitol, se prezintă elementele interfeței cu utilizatorul disponibile pentru Palm OS; sunt descrise funcțiile acestora și sunt date exemple pentru fiecare dintre ele.

Forme

O formă (*form*) pune la dispoziție un container, atât la nivel vizual, cât și programatic, pentru elementele interfeței cu utilizatorul. Formele conțin alte

elemente ale interfeței. O formă reprezintă, de obicei, un ecran al aplicației sau o cutie de dialog modală.

Fiecare aplicație trebuie să conțină măcar un *form*. Majoritatea *form*-urilor vor ocupa întreg ecranul, cu excepția dialogurilor, care pot să ocupe mai puțin din întreaga înălțime a ecranului, dar care se vor „întinde" pe toată lățimea acestuia.

O resursă de tip *form* se definește într-un fișier cu extensia *.rcp* și are următoarea sintaxă:

```
FORM ID <resourceID> AT (<Left> <Top> <Width> <Height>)
[MODAL]
[SAVEBEHIND|NOSAVEBEHIND]
[HELPID <resourceID>]
[DEFAULTBTNID <resourceID>]
[MENUID <resourceID>]
BEGIN
[TITLE "Form Title"]
<OBJECTS>
END
```

Parametrul `resourceID` reprezintă identificatorul formei.

Construcția `AT (<Left> <Top> <Width> <Height>)` apare în toate definițiile obiectelor de interfață și controlează localizarea și mărimea *form*-ului sau elementului de interfață. Parametrii `Left` și `Top` sunt coordonatele stânga și sus ale obiectului, iar `Width` și `Height` reprezintă lățimea și înălțimea obiectului în pixeli. De exemplu, următoarea linie produce un *form* standard de mărimea ecranului:

```
FORM ID 1000 AT (0 0 160 160)
```

Specificarea atributului `MODAL` produce un *form* având o margine de 2 pixeli, potrivit pentru a fi utilizat ca o căsuță de dialog modală. Următoarele linii creează o căsuță de dialog modală care ocupă întregul ecran:

```
FORM ID 2000 AT (2 2 156 156)
MODAL
```

Obținerea unui pointer la un *form* se realizează astfel:

```
FormType *form;
form = FrmGetActiveForm();
```

Alerte

Alertele constituie o facilitate pentru afișarea de dialoguri modale simple. Un dialog de tip *alert* este un tip special de *form* având o bară de titlu, un mesaj textual, unul sau mai multe butoane și o pictogramă (*icon*).

Un control de tip *alert* poate să aibă unul dintre următoarele tipuri:

- *Information* – un dialog informativ, folosit pentru a oferi utilizatorului informații de bază (de exemplu, faptul că o acțiune nu poate fi executată);
- *Confirmation* – întreabă utilizatorul în vederea introducerii de text sau pentru o anumită confirmare a unei acțiuni și pune la dispoziție un număr de butoane dintre care utilizatorul poate alege;
- *Warning* – trebuie folosit pentru a solicita confirmarea, atunci când utilizatorul a cerut executarea unei acțiuni care este potențial periculoasă. Diferența față de celelalte constă în faptul că acțiunea nu este reversibilă;
- *Error* – informează utilizatorul că ultima acțiune realizată a cauzat o eroare și nu a putut fi terminată.

Iată un exemplu de definire a unui *alert*:

```
ALERT ID Error1TooLong
ERROR
BEGIN
     TITLE "Numar prea mare"
     MESSAGE "Numarul poate avea maxim 4 cifre."
     BUTTONS "OK"
END
```

Butoane

O resursă de tip buton are următoarea sintaxă:

```
BUTTON <label> ID <resourceID> AT (<Left> <Top> <Width>
<Height>) [USABLE|NONUSABLE] [FRAME|NOFRAME]
[BOLDFRAME] [FONT <font number>]
```

Atributele `FRAME` și `NOFRAME` ale unui buton se exclud mutual. Dacă `FRAME` este setat, Palm OS desenează butonul cu o margine rectangulară. `NOFRAME` creează un buton fără această margine. Folosirea atributului `BOLDFRAME` generează un buton cu un cadru de 2 pixeli. Acest atribut nu are efect dacă butonul conține deja atributul `NOFRAME`.

Următorul exemplu definește un buton cu eticheta „Ajutor", având fontul standard, plasat în colțul din stânga jos al unui *form*:

```
BUTTON "Ajutor" ID 1000 AT (1 BOTTOM@159 AUTO AUTO) FONT 0
```

Check-box-uri

Un control de tip *check-box* se specifică astfel:

```
CHECKBOX <label> ID <resourceID> AT (<Left> <Top> <Width>
<Height>) [USABLE|NONUSABLE] [FONT <font number>]
```

```
[GROUP <group ID>] [CHECKED]
```

Unui grup i se poate atribui un *check-box* prin setarea atributului GROUP. Atributul CHECKED controlează comportarea controlului: dacă CHECKED este prezent, iar codul aplicației nu schimbă în mod explicit starea controlului, atunci butonul va fi afișat ca fiind selectat.

Următorul exemplu generează un *check-box* cu eticheta „Afișează detalii", care va fi implicit selectat și va fi afișat îngroșat:

```
CHECKBOX "Afiseaza detalii" ID 1001 AT (0 80 AUTO AUTO)
FONT 1 CHECKED
```

Câmpuri

Un câmp se definește astfel:

```
FIELD ID <resourceID> AT (<Left> <Top> <Width> <Height>)
[USABLE|NONUSABLE] [LEFTALIGN|RIGHTALIGN]
[FONT <font number>] [EDITABLE|NONEDITABLE] [UNDERLINED]
[SINGLELINE|MULTIPLELINES] [DYNAMICSIZE]
<MAXCHARS <number>> [AUTOSHIFT] [NUMERIC]
```

Atributele LEFTALIGN și RIGHTALIGN controlează modul în care textul este aranjat în interiorul câmpului, iar EDITABLE și NONEDITABLE specifică dacă un câmp acceptă intrări de la utilizator sau nu.

Atunci când apare UNDERLINED, sistemul desenează câmpul cu un contur gri. Atributele SINGLELINE și MULTIPLELINES controlează dacă un câmp este format dintr-o singură linie sau din mai multe. Includerea atributului DYNAMICSIZE cauzează lansarea unui eveniment fldHeightChanged, atunci când introducerea unui text de către utilizator produce apariția barei de defilare.

Pentru ca un câmp să funcționeze în mod optim, trebuie să setați atributul MAXCHARS pentru a stabili numărul maxim de caractere pe care le poate conține. Atributul NUMERIC este inclus pentru a restricționa valoarea unui câmp la numere.

Următorul exemplu definește un câmp editabil pe mai multe linii, care poate avea maximum 1KB (1024 caractere) de text:

```
FIELD ID 1003 AT (0 16 153 121) EDITABLE UNDERLINED
MULTIPLELINES MAXCHARS 1024 AUTOSHIFT
```

Bitmap-uri

Un control de tip *bitmap* (imagine) se definește astfel:

```
FORMBITMAP AT (<Left> <Top>) BITMAP <bitmapID>
[USABLE|NONUSABLE]
```

Pentru a include o resursă de tip *bitmap* într-un *form*, trebuie să o ataşaţi respectivului *form*. Atributul `BITMAP` trebuie să specifice ID-ul resursei *bitmap*, definit oriunde în fişierul *.rcp*.

Următorul exemplu plasează un *bitmap* cu ID-ul 100 în colţul din stânga sus al *form*-ului, chiar sub bara de titlu:

```
FORMBITMAP AT (0 16) BITMAP 100
```

Etichete

O etichetă este definită astfel:

```
LABEL <label text> ID <resourceID> AT (<Left> <Top>)
[USABLE|NONUSABLE] [FONT <font number>]
```

O etichetă nu are atributele `Height` şi `Width`. Mărimea sa este dependentă de textul conţinut. Se pot insera linii noi în interiorul şirului de caractere specificat prin folosirea caracterului special „*n*". Liniile lungi se pot continua pe următoarea linie prin folosirea caracterului *backslash* „\".

Următorul exemplu creează o etichetă în partea superioară a *form*-ului:

```
LABEL "Va rugam introduceti mai jos\n" \
"numele d-voastra:" AUTOID AT (0 16) FONT 1
```

Liste

O listă se defineşte astfel:

```
LIST <Item_1> <Item_2> ... ID <resourceID>
AT (<Left> <Top> <Width> <Height>) [USABLE|NONUSABLE]
[VISIBLEITEMS <number>] [FONT <font number>]
```

Atributul `VISIBLEITEMS` specifică numărul de elemente (*item*-uri) din listă care sunt vizibile la un moment dat.

Următorul exemplu generează o listă conţinând şase elemente, dintre care doar trei pot fi afişate la un moment dat.

```
LIST "Ton" "Cascaval" "Ardei" "Masline" "Ciuperci"
"Salam" ID 1005 AT (1 50 80 AUTO) VISIBLEITEMS 3
```

Butoane de execuţie (*Push*)

Un astfel de control se defineşte după cum urmează:

```
PUSHBUTTON <label> ID <resourceID>
AT (<Left> <Top> <Width> <Height>) [USABLE | NONUSABLE]
[FONT <font number>]
[GROUP <group ID>]
```

Aceste butoane pot fi grupate astfel încât doar unul dintre ele să poată fi apăsat la un anumit moment.

Următorul exemplu creează o linie de butoane de tip *push* pentru selectarea unui întreg cu valori între 1 şi 5:

```
PUSHBUTTON "1" ID 5001 AT (1 80 AUTO AUTO) GROUP 1
PUSHBUTTON "2" ID 5002 AT (PREVRIGHT+1 PREVTOP AUTO AUTO)
GROUP 1
PUSHBUTTON "3" ID 5003 AT (PREVRIGHT+1 PREVTOP AUTO AUTO)
GROUP 1
PUSHBUTTON "4" ID 5004 AT (PREVRIGHT+1 PREVTOP AUTO AUTO)
GROUP 1
PUSHBUTTON "5" ID 5005 AT (PREVRIGHT+1 PREVTOP AUTO AUTO)
GROUP 1
```

Butoane repetitive

Un control de acest tip este definit astfel:

```
REPEATBUTTON <label> ID <resourceID> AT (<Left> <Top>
<Width>
<Height>) [USABLE|NONUSABLE] [FRAME|NOFRAME]
[BOLDFRAME] [FONT <font number>]
```

Atributele pentru butoanele repetitive sunt similare celor de la butoanele normale, diferă doar modul în care se comportă în momentul rulării. Butoanele repetitive sunt ideale pentru săgeţile de *scroll*, deoarece utilizatorul poate să menţină *stylus*-ul pe acestea atunci când se deplasează să vizualizeze cantităţi mai mari de date.

Următorul exemplu generează o pereche de săgeţi de defilare verticală, plasate în colţul din dreapta jos al unui *form*.

```
REPEATBUTTON "\001" ID 2000 AT (147 144 13 8) FONT 5
NOFRAME
REPEATBUTTON "\002" ID 2001 AT (147 152 13 8) FONT 5
NOFRAME
```

Bare de defilare

O bară de defilare are următoarea definiţie:

```
SCROLLBAR ID <resourceID>
AT (<Left> <Top> <Width> <Height>)
[USABLE | NONUSABLE] VALUE <number>
MIN <number> MAX <number>
PAGESIZE <number>
```

O bară de defilare trebuie să aibă mereu o grosime de 7 pixeli. Folosirea cuvântului-cheie AUTO nu funcţionează, deoarece este utilizat pentru controale care au etichete text. Atributul VALUE setează localizarea iniţială a barei de defilare, iar PAGESIZE stabileşte cu câte unităţi se mută bara de defilare atunci când utilizatorul apasă deasupra sau sub aceasta, în interiorul controlului.

Trebuie setate următoarele atribute: VALUE, MIN, MAX şi PAGESIZE, altfel controlul nu va fi vizibil în interiorul formei.

Exemplul următor specifică o bară de defilare. Ea variază între 0 şi 100, carul de defilare începe la poziţia 40, mutându-se cu câte 10 unităţi la fiecare apăsare de buton.

```
SCROLLBAR ID 7000 AT (153 18 7 121) VALUE 40
MIN 0 MAX 100 PAGESIZE 10
```

Declanşatoare de selecţie

Un declanşator de selecţie se defineşte astfel:

```
SELECTORTRIGGER <label> ID <resourceID>
AT (<Left> <Top> <Width> <Height>) [USABLE | NONUSABLE]
[LEFTANCHOR | RIGHTANCHOR] [FONT <font number>]
```

Pentru a crea un declanşator de selecţie folosit ca un selector al datei, vom recurge la selectorul de dată Palm OS standard:

```
SELECTORTRIGGER "DAY MM/DD/YY" ID 1009
AT (56 34 AUTO AUTO) LEFTANCHOR
```

Tabele

Un tabel este definit astfel:

```
TABLE ID <resourceID> AT (<Left> <Top> <Width> <Height>)
ROWS <number> COLUMNS <number> COLUMNWIDTHS <width 1>
<width 2> ...
```

Atributele ROWS şi COLUMNS conţin numărul de linii şi, respectiv, de coloane ale unui tabel. Pentru a seta lăţimile coloanelor, după atributul COLUMNWIDTHS trebuie specificate valori separate prin spaţii.

Următorul exemplu generează un tabel cu cinci coloane. Înălţimea stabilită va permite afişarea a 11 linii.

```
TABLE ID 1010 AT (0 18 160 121) ROWS 11 COLUMNS 5
COLUMNWIDTHS 12 10 96 27 24
```

2.5 Lucrul cu bazele de date Palm OS

Prezentare a managerului de date (*Data Manager*)

O bază de date Palm OS reprezintă o listă de zone din cadrul memoriei RAM a *handheld*-ului, împreună cu o informaţie antet (*header*) care descrie baza de date însăşi.

Pentru a permite o mai bună utilizare a memoriei, fiecare înregistrare din baza de date poate să fie inclusă în orice *heap* de memorie de pe *handheld*, alături de înregistrări provenind din alte aplicaţii. Această „întrepătrundere" a înregistrărilor permite sistemului să organizeze datele în orice mod doreşte, pentru ca unele înregistrări să fie mutate dacă spaţiul pe care îl ocupă trebuie folosit să stocheze o înregistrare de lungime mai mare (necesitând spaţiu de memorare contiguu).

În lista de înregistrări a bazei de date, fiecare înregistrare este disponibilă printr-un *LocalID*, un deplasament (*offset*) de memorie, având ca punct de referinţă adresa de început a cardului care conţine înregistrarea. Stocarea adreselor de memorie ca *LocalID*-uri, în loc de pointeri la înregistrări, permite sistemului să realoce un întreg card de memorie, fără a fi nevoie să modifice locaţia fiecărei înregistrări stocate în lista de înregistrări a bazei de date.

Sistemul poate realoca date la cerere în cadrul memoriei *handheld*-ului, astfel că *LocalID*-urile sunt valide numai între momentul în care deschideţi o bază de date şi până o închideţi. Salvarea unui *LocalID* corespunzător unei înregistrări (sau salvarea doar a *header*-ului unei baze de date) este inutilă, deoarece în momentul în care aplicaţia a închis o bază de date, sistemul poate să mute datele oriunde, în acest mod invalidând *LocalID*-ul pe care l-aţi salvat.

Chiar dacă lista de înregistrări a bazei de date foloseşte *LocalID*-uri pentru a organiza înregistrările, codul din aplicaţii solicită înregistrările pe baza indexului din lista de înregistrări.

Managerul de baze de date (*Data Manager*) Palm OS utilizează indexul pentru a găsi *LocalID*-ul înregistrării cerute, converteşte *LocalID*-ul la un *handle* bazat pe cardul pe care se află *header*-ul bazei de date, iar apoi returnează *handler*-ul aplicaţiei.

Crearea, deschiderea şi închiderea bazelor de date

Bazele de date de pe un *handheld* Palm OS sunt asemănătoare fişierelor de pe un computer *desktop*, cu excepţia că primele se află în RAM. Folosind funcţii din API-ul Palm OS, aplicaţia poate crea şi şterge bazele de date, le poate deschide şi închide, asemănător modului în care un sistem *desktop* utilizează fişierele. Punctul în care bazele de date Palm OS diferă de fişierele de pe un calculator *desktop* constă în citirea datelor şi scrierea acestora în baza de date.

Pentru a crea o bază de date, trebuie apelată funcţia `DmCreateDatabase()`. Prototipul acestei funcţii este următorul:

```
Err DmCreateDatabase (UInt16 cardNo, const Char * nameP,
UInt32 creator, UInt32 type, Boolean resDB)
```

Parametrul `cardNo` reprezintă cardul de memorie pe care doriţi să creaţi baza de date. Actualmente, nici un *handheld* Palm OS nu are mai mult decât un card de memorie, astfel încât puteţi să daţi valoarea 0 acestui parametru pentru a indica faptul că se foloseşte primul card de pe dispozitiv.

Parametrul `nameP` acceptă un şir care să fie folosit ca un nume pentru baza de date. Doar 32 de octeţi sunt disponibili în *header*-ul bazei de date pentru a stoca acest nume, care include 1 octet pentru terminare, astfel că numele bazei de date poate fi de maximum 31 de caractere lungime.

În general, o aplicaţie trebuie să verifice existenţa bazei sale de date în rutina `StartApplication()`. Dacă baza de date nu există, `StartApplication()` poate apela `DmCreateDatabase()` pentru a crea o nouă bază de date pentru aplicaţie.

Următorul fragment din funcţia `StartApplication()` a aplicaţiei *Librarian* verifică dacă există baza de date a aplicaţiei şi, dacă nu, creează una nouă:

```
#define libDBName "LibrarianDB-LFlb"
#define libDBType 'DATA'
#define libCreatorID 'LFlb'

Err error = 0;
UInt16 mode;

...

/* Gaseste baza de date pentru aplicatie */
gLibDB = DmOpenDatabaseByTypeCreator(libDBType,
libCreatorID, mode);

/* Daca aceasta nu exista, o creeaza */
if (! gLibDB) {
  error = DmCreateDatabase(0, libDBName, libCreatorID,
            libDBType, false);

  /* Daca apare o eroare, atunci returneaza eroarea */
  if (error)
    return error;

  /* Incearca din nou deschiderea bazei de date */
  gLibDB = DmOpenDatabaseByTypeCreator(libDBType,
             libCreatorID, mode);
```

```
  if (! gLibDB)
    return DmGetLastErr();

  ...
}
```

Rutina `StartApplication()` verifică inițial dacă baza sa de date *Librarian* există, încercând să o deschidă cu `DmOpenDatabaseByTypeCreator()`, care returnează fie o referință la baza de date deschisă, fie valoarea 0. Dacă baza de date nu există, se apelează `DmCreateDatabase()` cu numele bazei de date a aplicației, ID-ul creatorului și tipul *DATA* pentru a crea noua bază de date. Apoi `StartApplication()` deschide nou-creata bază de date, astfel încât *Librarian* va avea acces la ea.

Dacă deschiderea acestei baze de date secundare eșuează, se apelează funcția `DmGetLastErr()` pentru a primi codul ultimei erori detectate de managerul de date și se trimite acest cod spre rutina `PilotMain()`.

Funcția `DmOpenDatabaseByTypeCreator()` deschide o bază de date, furnizându-se tipul și ID-ul creatorului bazei de date, și returnează o referință la baza de date deschisă, dacă această acțiune se realizează cu succes. Tipul de date folosit pentru referința la o bază de date deschisă și pentru variabila globală `gLibDB` din exemplul anterior este *DmOpenRef*.

Un al treilea parametru pentru `DmOpenDatabaseByTypeCreator()` stabilește modul în care baza de date ar trebui deschisă. Palm OS furnizează un număr de constante (prezentate în tabelul următor) pentru a defini modurile de acces la baza de date.

Mod	Descriere
`dmModeReadWrite`	Acces de citire/scriere.
`dmModeReadOnly`	Acces numai pentru citire.
`dmModeWriteOnly`	Acces numai pentru scriere.
`dmModeLeaveOpen`	Lasă baza deschisă și după închiderea aplicației.
`dmModeExclusive`	Exclude alte aplicații pentru deschiderea bazei de date.
`dmModeShowSecret`	Listează înregistrările din această bază de date care sunt marcate ca fiind private.

Constante de acces la baza de date

Atunci când aplicația a terminat lucrul cu o bază de date, trebuie să apeleze funcția `DmCloseDatabase()` pentru a o închide. Această funcție primește ca parametru o referință de tipul *DmOpenRef* la o bază de date deschisă:

```
DmCloseDatabase(gLibDB);
```

Funcţia `StopApplication()` a unei aplicaţii este un loc potrivit pentru a apela `DmCloseDatabase()`, chiar imediat după orice cod care realizează orice altă „curăţare" a memoriei folosite de aplicaţie.

Citirea dintr-o înregistrare

Citirea dintr-o înregistrare este foarte simplă. Deşi înregistrările sunt protejate la scriere, ele se află în RAM. De aceea, puteţi să preluaţi o înregistrare din baza de date, să o blocaţi şi apoi să citiţi din ea.

Iată un exemplu:

```
/* obtine inregistrarea */
VoidHand myRecord = DmQueryRecord(gDB, recordNumber);

/* blocheaza inregistrarea */
StructType *s = MemHandleLock(myRecord);

/* realizeaza actiuni cu inregistrarea */
Actiune(s->field);

/* deblocheaza inregistrarea */
MemHandleUnlock(myRecord);
```

Apelul funcţiei `DmQueryRecord()` returnează o înregistrare care este *read-only* (nu poate fi modificată, ci doar consultată).

Iterarea înregistrărilor dintr-o bază de date

Chiar dacă doriţi numai elementele dintr-o categorie anume sau toate înregistrările, trebuie să folosiţi apeluri pentru categoria respectivă. Aceste apeluri ignoră înregistrările şterse sau arhivate şi înregistrările private – dacă baza de date nu este deschisă cu `dmModeShowSecret()`. Următorul exemplu de cod iterează toate înregistrările:

```
UInt theCategory = dmAllCategories;
  /* poate fi o categorie specifica */
UInt totalItems =
  DmNumRecordsInCategory(gDB, theCategory);
UInt i;
UInt recordNum = 0;
for (i = 0; i < totalItems; i++) {
  VoidHand recordH = DmQueryNextInCategory
    (gDB, &recordNum, theCategory);
    /* la acest punct, recordNum contine numarul
       inregistrarii dorite */
}
```

Lucrul cu înregistrări mari

Cantitatea maximă de date pe care o înregistrare poate să o înmagazineze este puțin mai mică de 64 KB. Dacă aveți date de dimensiuni mai mari cu care doriți să lucrați, există mai multe posibilități să realizați acest lucru – pentru documentații suplimentare, a se consulta *http://www.palmsource.com/developers/*.

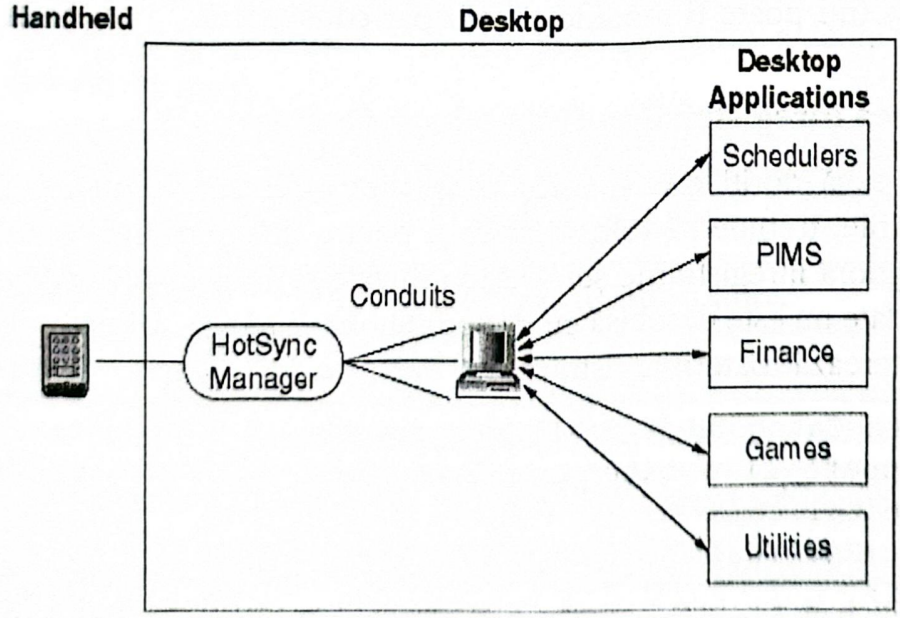
3. Sincronizarea cu calculatorul: proiectarea conduitelor

3.1 Preliminarii

Conectând dispozitivul Palm OS la calculator, se pot executa următoarele operații:

- sincronizarea datelor stocate pe *handheld* cu datele stocate pe *desktop*;
- realizarea de copii de siguranță (*backup*) – pe *desktop* – ale datelor aflate pe *handheld*;
- instalarea bazelor de date ale aplicațiilor *handheld* care au fost stocate pe *desktop*.

În figura de mai jos se pot observa relațiile dintre componente:



Relațiile dintre *handheld* și *desktop*

HotSync Manager este o aplicație *desktop* care realizează managementul comunicării cu dispozitivul *handheld*. *HotSync Manager* folosește un API pentru comunicații, numit *Sync Manager* API, pentru a trata trimiterea și pentru recepționarea de octeți de la și spre *handheld*, fapt ce determină ca *HotSync Manager* și *Sync Manager* API să fie independente de tipul conexiunii.

Tipurile de conexiuni pentru comunicare includ: conexiunea directă prin cablu serial, conexiunea prin infraroşu, conexiunea via modem şi conexiunea reţea.

3.2 Prezentarea conduitelor

O conduită este un modul *plug-in* pentru *HotSync Manager* care transferă un anumit tip de date între *handheld* şi *desktop*. Conduitele nu trebuie să necesite intervenţii din partea utilizatorului.

Unele aplicaţii *handheld* creează sau folosesc date care pot fi partajate cu datele de pe o platformă *desktop*. De exemplu, aplicaţia *Date Book* realizează schimbul de date *Date Book* cu software-ul *Palm Desktop* care rulează pe platforma *desktop*.

Alte aplicaţii *handheld* generează date care sunt salvate apoi pe *desktop*. O conduită de *backup* copie datele de pe Palm pe *desktop*.

Instrumente de dezvoltare a conduitelor

Pentru a permite dezvoltatorilor să creeze conduite, Palm oferă instrumente de tip CDK (*Conduit Development Kits*), pentru Windows şi Macintosh.

Instrumentul CDK pentru Windows conţine trei module principale:

- *C/C++ Sync Suite* – componentele CDK pentru Windows folosite pentru a crea o conduită bazată pe un API în limbajul C. Include API-ul, clasele *framework* C++, bibliotecile C, exemple, documentaţie şi utilitare suplimentare.
- *COM Sync Suite* – componentele CDK pentru Windows folosite pentru a crea conduite bazate pe standardul COM. Include obiecte/metode/proprietăţi COM, interfaţa *COM Sync* pentru a dezvolta conduita bazată pe COM cu *HotSync Manager*, exemple, documentaţie, utilitare şi *COM Sync Installer*.
- *JSync Suite* – componentele folosite pentru a crea o conduită în limbajul Java. Include pachetele, clasele, exemplele, documentaţiile, interfaţa *JSync* pentru dezvoltarea conduitelor în Java, alte utilitare şi *JSync Installer*.

Tipuri de conduite

Conduitele pot executa o multitudine de acţiuni, dar se încadrează, în principal, în trei tipuri majore, care sunt prezentate în tabelul următor:

Tip	Descriere	Explicații
Install conduit (conduită de instalare)	Instalează aplicații Palm OS și baze de date în memoria principală a *handheld*-ului sau fișiere pe carduri de extensie. *HotSync Manager* include o astfel de conduită implicită.	În Windows, *HotSync Manager* rulează conduite de instalare atât înainte, cât și după ce rulează conduitele de sincronizare (*synchronization conduits*).
Backup conduit (conduită de salvare a copiilor de siguranță)	Copie pe *desktop* orice bază de date *handheld* și aplicații care au setat bitul de copiere (*backup*). *HotSync Manager* include o *backup conduit* implicită. *Backup conduits* sunt proiectate pentru bazele de date *handheld* și aplicațiile care nu au conduite de sincronizare asociate lor.	*HotSync Manager* rulează *backup conduit* implicită după ce invocă conduitele de sincronizare.
Synchronization conduit (conduită de sincronizare)	Realizează orice acțiune pe care dezvoltatorul acesteia o dorește, tipic sincronizând bazele de date de pe *handheld* cu părțile corespunzatoare de pe *desktop*.	În Windows, *HotSync Manager* rulează conduitele de sincronizare înregistrate numai dacă există o bază de date de tipul *apply* pe *handheld*.

Tipuri de conduite

Funcții de bază ale conduitelor generice

Ca orice altă conduită, o conduită generică începe sincronizarea datelor în funcția `OpenConduit()`. Această funcție instanțiază subclasa proprie clasei `Csynchronizer`, iar apoi invocă metoda `Perform()` a obiectului de sincronizare. După ce acest obiect și-a terminat acțiunea, `OpenConduit()` îl distruge și are loc ieșirea din funcție. Iată un exemplu pentru această funcție:

```
ExportFunc long OpenConduit (PROGRESSFN
pFn,CSyncProperties& Props) {
long retval = -1;

if (pFn) {
      CLibCondSync* pGeneric;
      pGeneric = new CLibCondSync (rProps,
        GENERIC_FLAG_CATEGORY_SUPPORTED |
        GENERIC_FLAG_APPINFO_SUPPORTED);
    if (pGeneric) {
        retval = pGeneric->Perform();
        delete pGeneric;
    }
}
return(retval);
}
```

Următorul exemplu prezintă o declarare a unei clase de înregistrări care conține un singur șir de caractere pentru date:

```
class CMyConduitRecord
{
public:
CMyConduitRecord(void); // constructor
~CMyConduitRecord();    // destructor

public:                  // date-membre publice
DWORD m_dwRecordID;
DWORD m_dwStatus;
DWORD m_dwPosition;
CPString m_csData;
DWORD m_dwPrivate;
DWORD m_dwCategoryID;
public:                  // metode publice
void Reset(void);
};
```

Iată implementarea clasei `CMyConduitRecord` declarate mai sus:

```
CMyConduitRecord::CMyConduitRecord (void)
{
Reset();
}

CMyConduitRecord::~CMyConduitRecord() {}

void CMyConduitRecord::Reset(void)
{
m_dwRecordID = 0;
m_dwStatus = 0;
m_dwPosition = 0;
m_dwPrivate = 0;
m_dwCategoryID = 0;
m_csData.Empty();
}
```

4. Aplicația practică *PalmNotes*

4.1 Introducere

Pentru a evidenția principalele metode de construire a aplicațiilor pentru platforma Palm OS, a fost realizată aplicația *PalmNotes*, reprezentând un catalog electronic prin intermediul căruia se pot acorda note (punctaje)

elevilor/studenților înscriși la laboratoare sau seminarii, pentru diferite discipline. Această aplicație devine un sistem flexibil prin intermediul căruia se poate realiza o gestiune completă a situației școlare la diverse materii.

Astfel, aplicația reprezintă un ansamblu de două componente principale, implementate pe două platforme eterogene între care se realizează o interacțiune la nivel de schimb de date. Acestea sunt: o parte implementată pe un dispozitiv mobil de tip *handheld*, realizată în limbajul C, și cealaltă dezvoltată pe o platformă *desktop*, pentru implementare recurgându-se la ASP.NET. Legăturile dintre cele două componente se realizează prin intermediul unui protocol de comunicație dezvoltat integral în C.

Implementarea celor două componente diferă total. Astfel, pe platforma *handheld* au fost urmărite aspecte precum optimizarea memoriei folosite, rapiditate în navigare și simplitate, existând limitări hardware precum memoria și capacitatea de calcul redusă. În schimb, în ceea ce privește aplicația *desktop*, aceasta are o funcționalitate extinsă, permițând acțiuni și operații care nu au fost implementate pe *handheld*.

4.2 Proiectarea aplicației

Aplicația *PalmNotes* a fost gândită să satisfacă în primul rând cerințele profesorului – posesor de dispozitiv mobil de tip *handheld*; de asemenea, și un utilizator care nu dispune de un astfel de dispozitiv mobil poate să utilizeze la fel de bine aplicația pe platforma *desktop*.

Cerințele aplicației ar putea fi rezumate de următoarele aserțiuni:

- sistemul trebuie să fie ușor de utilizat;
- sistemul trebuie să permită o sincronizare facilă a notelor între dispozitivul mobil și *desktop*;
- sistemul trebuie să ofere acces în rețea sau de pe Internet;
- sistemul trebuie să fie rapid și să permită în timp dezvoltarea de module adiționale.

Din punctul de vedere al funcționalității celor două module ale aplicației, diferența dintre acestea se poate observa în continuare.

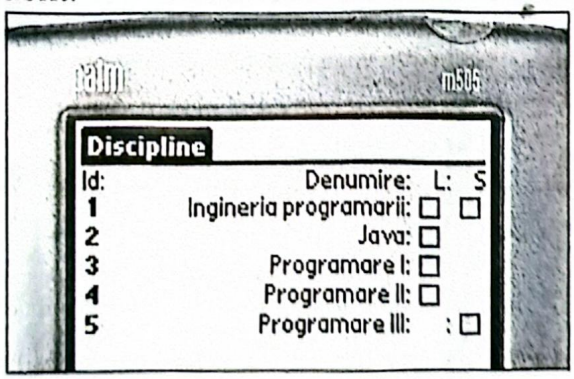
Se observă faptul că funcționalitatea aplicației *handheld* este mai restrânsă, scopul acesteia fiind de a realiza introducerea de note de către profesor, dar și managementul prezenței studentului la oră. Aceste acțiuni se realizează instantaneu pe partea *handheld*, care se bucură de avantajul mobilității.

Operație	Modul *handheld*	Modul *desktop*
Selectarea disciplinei	Implementată	Implementată
Selectarea laboratorului	Implementată	Implementată
Selectarea grupei	Implementată	Implementată
Selectarea studentului	Implementată	Implementată
Editarea notelor studentului	Implementată	Implementată
Editarea prezenței studentui	Implementată	Implementată
Adăugare de discipline, grupe, laboratoare /seminarii, studenți	Neimplementată	Implementată
Editare de date discipline, grupe, laboratoare/seminarii, studenți	Neimplementată	Implementată
Realizare de rapoarte cu notele studenților	Neimplementată	Implementată

La acest nivel, nu ar avea rost să se implementeze operațiuni detaliate de editare (precum schimbarea numelui unei discipline sau adăugarea unui student), întrucât acestea nu se realizează frecvent. Menținerea unor module care să realizeze aceste acțiuni se dovedește inutilă pe un sistem în care memoria consumată este foarte importantă. În schimb, în cadrul aplicației *desktop*, aceste module sunt implementate și se pot utiliza foarte eficient.

Prezentarea modulelor și a funcționalităților platformei *handheld*

În continuare, se vor prezenta acțiunile aplicației Palm OS, urmând o parcurgere a formelor acesteia pentru a exemplifica modalitatea în care poate fi notat un student.

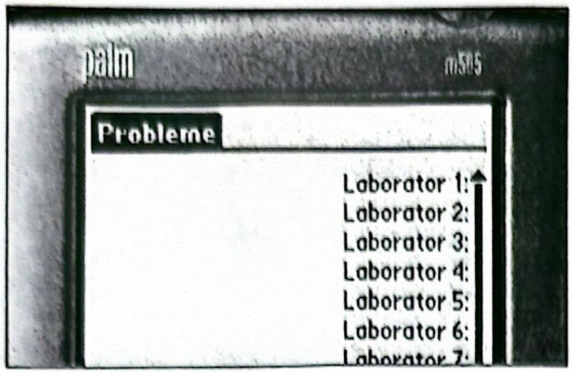


Vizualizarea disciplinelor, cu selecția laboratoarelor sau seminariilor

Selectarea unei discipline se realizează prin alegerea tipului de activitate pentru disciplina respectivă. Astfel, se poate bifa fie „Laborator" (marcat cu „L" în imaginea de mai sus), fie „Seminar" (marcat cu „S").

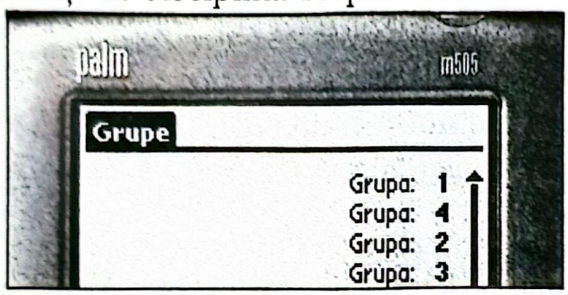
După selectarea unei discipline, urmează stabilirea laboratorului sau seminarului dorit. Aceasta se realizează în cadrul *form*-ului „Probleme". Se

poate reveni în cadrul meniului anterior prin apăsarea butonului „Înapoi" din aceeași secțiune.



***Form*-ul „Probleme"**

Selectarea unei grupe urmează selectării tipului de activitate pentru disciplina respectivă. Astfel, pe baza datelor din tabelul cu studenți existent în baza de date, are loc popularea unei liste cu toate grupele curente pentru studenții care sunt înscriși la disciplina respectivă.



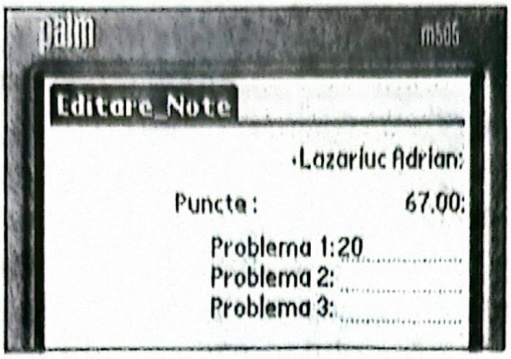
Selectarea grupelor

Urmează editarea prezenței studentului la cursul/laboratorul selectat. Căsuțele bifate din imaginea de mai jos indică faptul că studentul este prezent la laboratorul/seminarul selectat. Trecerea în secțiunea cu note se realizează prin *click* pe numele studentului.

Realizarea prezenței

Notarea studentului se realizează în cadrul *form*-ului „Editare note". Aici se vor afișa problemele existente la disciplina respectivă, pentru laboratorul

sau seminarul care a fost selectat. Se va vizualiza, de asemenea, punctajul studentului la disciplina respectivă, iar problemele la care a fost notat deja vor avea trecute punctajele în regiunea de introducere a notelor, ca în imaginea de mai jos.

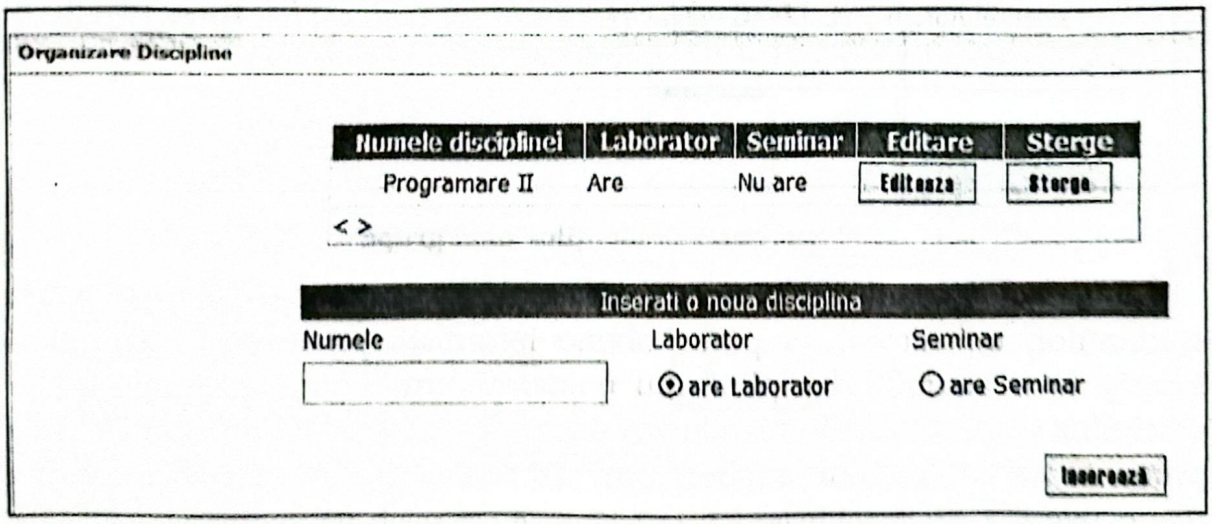


Editarea notelor unui student

Prezentarea modulelor şi a funcţionalităţilor platformei *desktop*

Aplicaţia *desktop* se prezintă sub forma unui sit Web, accesul la acesta făcându-se pe bază de nume de utilizator şi parolă. O dată autentificat cu succes, utilizatorului i se permite accesul la secţiunile de organizare a sitului.

Pagina pentru organizarea disciplinelor pune la dispoziţia utilizatorilor un sistem în cadrul căruia pot executa diferite acţiuni privind disciplina dorită. Astfel, aceştia pot să adauge, să editeze sau să şteargă o disciplină; în plus, ei pot să-şi organizeze apoi laboratoarele sau seminariile, ca în imaginea de mai jos.



Fragment din pagina de organizare a disciplinei

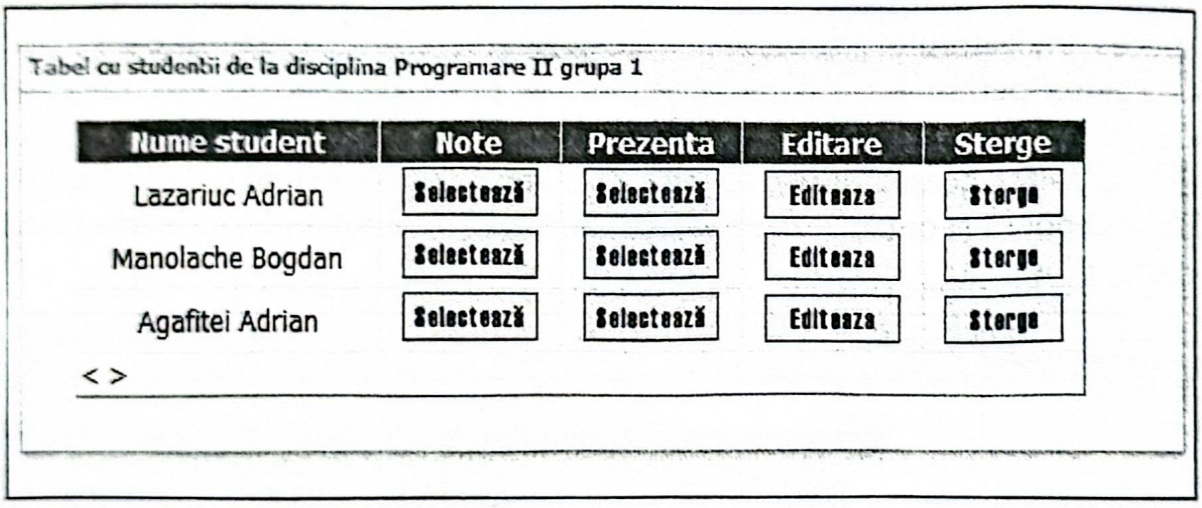
De notat aici faptul că disciplinele trebuie să primească nume diferite, la adăugarea unei noi discipline realizându-se automat validarea numelui acesteia. Totodată, trebuie selectat tipul de activitate pentru care se va

realiza managementul datelor, acesta fiind selectat implicit ca fiind de tip „Laborator".

Operația de ștergere permite eliminarea ireversibilă a unei discipline din baza de date. Astfel, înainte de a se lua decizia de a se șterge o disciplină, utilizatorul trebuie să știe că din baza de date se vor elimina toate înregistrările care sunt în relație cu disciplina respectivă (note, prezența la disciplina respectivă etc.).

Pentru a avea acces la modulul de adăugare a notelor și prezențelor unui student, trebuie să selectați – din pagina care se deschide în urma accesării meniului *Organizare Studenți* – grupa la care acesta este înscris. Apoi, veți avea acces la toți studenții înscriși la disciplina selectată anterior, provenind din grupa respectivă. În exemplul următor, se observă listați studenții înscriși la disciplina *Programare II* – grupa 1. Accesul la note se va realiza prin apăsarea butoanelor „Selectează" din dreptul fiecărui student.

Totodată, în această pagină puteți să ștergeți înregistrările care conțin date pentru un anumit student sau să editați numele său, în cazul în care acesta nu ar fi fost trecut inițial în mod corect.



Vizualizarea studenților unei grupe

În cadrul paginii de generare de rapoarte cu notele și activitatea studenților, utilizatorul va putea obține informații privitoare la studenții înscriși la o anumită disciplină sau numai pentru o anumită grupă de la disciplina respectivă. Informațiile vor prezenta punctajul total și numărul de prezențe ale studentului la laboratorul sau seminarul din cadrul disciplinei respective.

Arhitectura bazelor de date

- *Datele despre discipline*

Datele referitoare la discipline sunt stocate în tabela *asdiscipline*. Această tabelă conține un câmp în care se salvează numele disciplinei, un câmp în

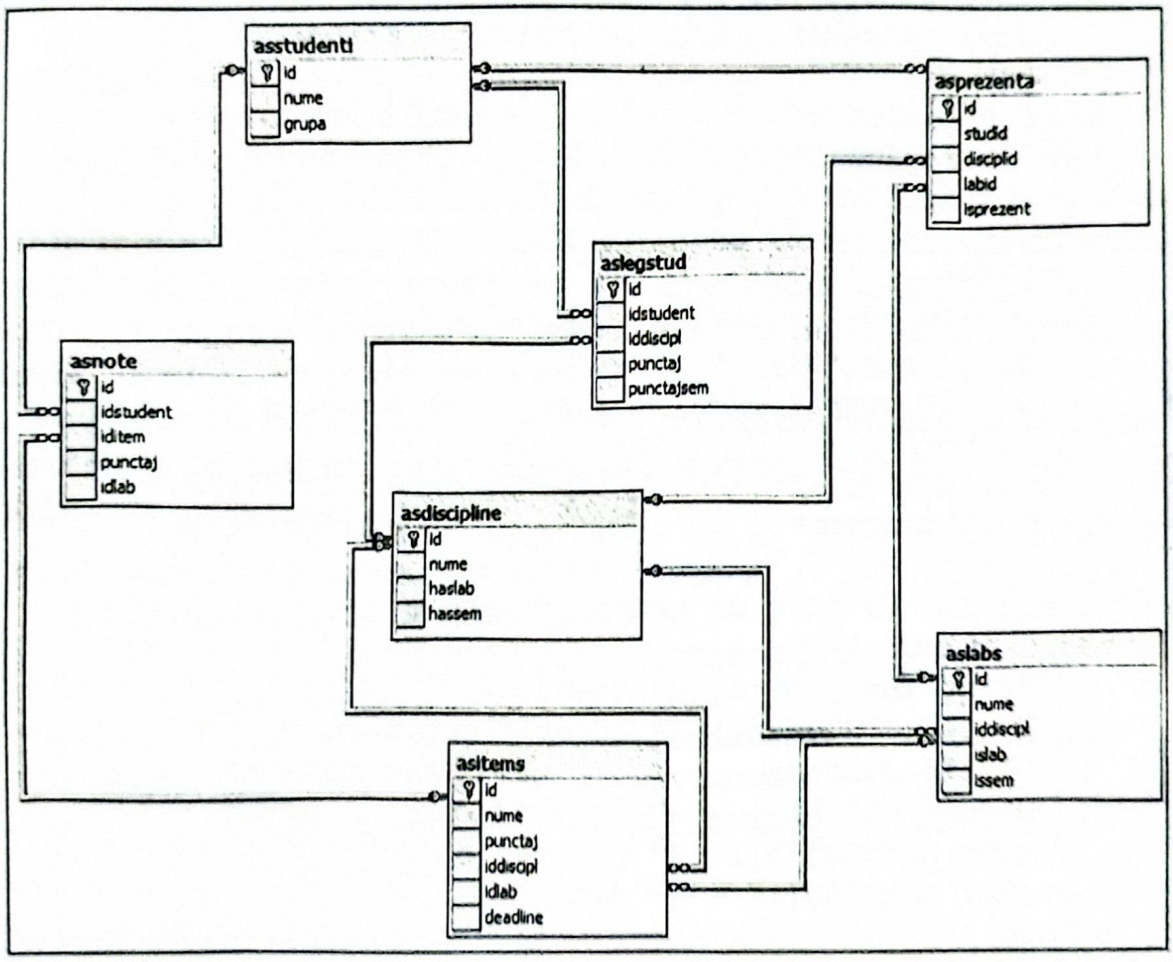
care se salvează un *flag* indicând faptul că disciplina respectivă are laborator şi încă un câmp care conţine un *flag* precizând dacă disciplina are asociate ore de seminar.

- *Datele despre studenţi*

Aceste date sunt păstrate în tabela *asstudenti*, cuprinzând numele şi grupa la care se află înscris fiecare student. Identificarea studenţilor se realizează prin intermediul ID-ului, care este, de altfel, cheie primară şi există în fiecare tabelă din baza de date.

- *Legăturile dintre datele studenţilor*

Acestea se menţin prin intermediul unor tabele auxiliare. Tabela care realizează legătura dintre studenţi şi discipline este numită *aslegstud*. Celelalte tabele menţin informaţii despre note, prezenţă, laboratoare, probleme, iar relaţiile dintre acestea se pot observa în diagrama de mai jos.



Reprezentarea tabelelor din baza de date şi a relaţiilor dintre acestea

Această structură este reprezentarea tabelelor bazei de date în MS SQL Server. Pentru aplicaţia de pe *handheld*, datele vor fi organizate prin intermediul unor structuri diferite:

```
/* structura memorand datele despre discipline */
typedef struct {
```

```
        UInt16 ID;      /* identificator unic */
        char nume[50]; /* numele disciplinei */
        UInt16 hasLab; /* flag - are laborator */
        UInt16 hasSem; /* flag - are seminar */
} DBDiscipline;
typedef DBDiscipline                    DBDisciplineType;
typedef DBDiscipline*                   DBDisciplinePtr;

/* structura in care se salveaza datele
   despre legaturile studenti-discipline */
typedef struct {
        UInt16 ID;          /* identificator */
        UInt16 IDStudent; /* identificator student */
        UInt16 IDDiscipl; /* identificator disciplina */
        char punctaj[10]; /* punctaj obtinut */
        char punctajSem[10]; /* punctaj seminar */
} DBLegStuds;
typedef DBLegStuds                      DBLegStudsType;
typedef DBLegStuds*                     DBLegStudsPtr;

/* structura in care se salveaza datele despre studenti */
typedef struct {
        UInt16 ID;      /* identificator */
        char nume[50]; /* nume student  */
        UInt16 grupa;   /* grupa din care provine */
} DBStudents;
typedef DBStudents                      DBStudentsType;
typedef DBStudents*                     DBStudentsPtr;

/* structura in care se salveaza datele despre note */
typedef struct {
        //int ID;
        UInt16 IDStudent; /* identificator student */
        UInt16 IDItem;     /* identificator nota */
        UInt16 IDLab;      /* identificator laborator */
        char punctaj[5];   /* punctaj obtinut */
        char time[50];     /* timp */
} DBNote;
typedef DBNote                          DBNoteType;
typedef DBNote*                         DBNotePtr;

/* structura in care se salveaza datele despre prezenta */
typedef struct {
        UInt16 IDStudent; /* identificator student  */
        UInt16 IDDiscipl; /* identificator discipl. */
        UInt16 IDLab;     /* identificator laborator */
```

```
        UInt16 ePrezent;   /* flag de prezenta */
} DBPrez;
typedef DBPrez                          DBPrezType;
typedef DBPrez*                         DBPrezPtr;

/* structura care stocheaza datele despre laboratoare */
typedef struct {
        UInt16 ID;          /* identificator */
        UInt16 isLab;       /* flag - este laborator */
        UInt16 isSem;       /* flag - este seminar */
        char nume[50];      /* nume */
        UInt16 IDDiscipl; /* identifica disciplina */
} DBLabs;
typedef DBLabs                          DBLabsType;
typedef DBLabs*                         DBLabsPtr;
```

Securitatea aplicației

În cele ce urmează vom discuta numai despre asigurarea securității aplicației *desktop*. Securitatea este îndeplinită prin autentificarea la nivel de formular oferită de platforma ASP.NET. Acest nivel de securitate funcționează în modul următor: când un utilizator este autentificat, se creează un *cookie* pentru a menține informațiile de autentificare corespunzătoare utilizatorului curent pentru sesiunea curentă.

Autentificarea bazată pe formulare este activată de introducerea în fișierul *Web.config* (fișierul de configurare a unei aplicații Web ASP.NET) a unei intrări care arată astfel:

```
<authentication mode="Forms">
  <forms loginUrl="login.aspx">
    <!-- specifica formularul de conectare -->
        <credentials passwordFormat="Clear">
              <!-- lista utilizatorilor autentificati
                   prin nume si parola -->
              <user name="admin" password="12345" />
        </credentials>
  </forms>
</authentication>
```

Atributul `loginUrl` specifică pagina de intrare în cont a utilizatorului, pagina spre care va trebui să fie redirecționat vizitatorul de fiecare dată când dorește să acceseze resursele aplicației pentru care nu este permis accesul anonim. Atenție! Parola este furnizată în clar, nefiind criptată.

De exemplu, aplicația *PalmNotes* are configurate toate paginile Web (cu excepția paginii de conectare *login.aspx*), astfel încât accesul să fie

restricționat utilizatorilor neautentificați. Utilizatorii pentru care accesul nu este permis se declară în aplicație într-o secțiune precum următoarea:

```
<authorization>
   <deny users="?" />
</authorization>
```

Semnul „?" este utilizat pentru a specifica utilizatorii anonimi (adică neautentificați).

Când un vizitator dorește să acceseze oricare dintre aceste pagini, mecanismul de securitate bazat pe formulare ASP.NET îl redirecționează automat către pagina principală și va continua acest lucru până când utilizatorul se autentifică prin nume și parolă.

Funcțiile principale ale aplicației de pe dispozitivul Palm OS

```
/* Elibereaza forma principala */
      void MainFormDeinit();

/* Elibereaza forma responsabila cu prezentarea laboratorului */
      void FormLabsDeinit();

/* Elibereaza forma folosita la prezentarea grupelor */
      void FormGrupeDeinit();

/* Elibereaza forma utilizata la prezentarea studentului */
      void FormStudentsDeinit();

/* Elibereaza forma responsabila cu prezentarea notelor */
      void FormNoteDeinit();

/* Initializeaza forma responsabila cu prezentarea laboratorului */
      void FormLabsInit(FormPtr form);

/* Initializeaza forma folosita la prezentarea grupelor */
      void FormGrupeInit(FormPtr form);

/* Initializeaza forma utilizata la prezentarea studentului */
      void FormStudentsInit(FormPtr form);

/* Initializeaza forma responsabila cu prezentarea notelor */
      void FormNoteInit(FormPtr form);

/* Converteste un double la string */
      void doubleToStr9(char* str, double flpNumber,
                        int numFractDigits);
```

```
/* Seteaza bitul de backup al aplicatiei */
      void SetDBBackupBit(DmOpenRef dbP);

/* Functie de preluare a datelor */
      void fnGetData(void* table, Int16 row,
       UInt16 column, Boolean editable,
       MemHandle* textHP, UInt16* textOffset,
       UInt16* textAllocSize, FieldPtr fld);

/* Converteste un string la double */
      double strToDouble(char* str);

/* Salveaza un camp la o adresa in baza de date. Este
   apelata de catre obiectul de tip tabel ca o rutina
   de callback atunci cand vrea sa salveze un item. */
      Boolean PrvEditSaveRecordField (void* table,
              Int16 row, Int16 column);

/* Returneaza un pointer la un camp al unei inregistrari
   de la o anumita adresa. Ea este apelata de obiectul tabel
   atunci cand se doreste a se afisa sau
   edita un camp. */
      Err PrvEditGetRecordField (void* table, Int16 row,
       Int16 column, Boolean editing, MemHandle* textH,
       Int16* textOffset, Int16 *textAllocSize,
       FieldPtr fld);

/* Seteaza valoarea unui camp */
      Err Set_field_value(UInt16 objectID, char* result);

/* Deschide tabelul cu discipline in baza de date */
      Err MemoAppInfoInit(DmOpenRef dbP);

/* Deschide tabelul cu laboratoare in baza de date */
      Err MemoAppInfoInitLabs(DmOpenRef dbP);

/* Deschide tabelul cu legaturi in baza de date */
      Err MemoAppInfoInitLegStuds(DmOpenRef dbP);

/* Deschide tabelul cu studenti in baza de date */
      Err MemoAppInfoInitStudents(DmOpenRef dbP);

/* Realizeaza setari ale inregistrarii din baza de date */
      Err MemoAppInfoInitRecords(DmOpenRef dbP);
```

```
/* Deschide tabelul cu note in baza de date */
    Err MemoAppInfoInitNote(DmOpenRef dbP);

/* Creeaza inregistrari implicite cu discipline in baza
   de date la instalarea aplicatiei pe Palm;
   aceste inregistrari sunt scrise pentru a realiza
   testarea aplicatiei si vor fi sterse ulterior,
   atunci cand se realizeaza sincronizarea */
    Err functie_baza();

/* Creeaza inregistrari implicite cu probleme in baza
   de date la instalarea aplicatiei pe Palm */
    Err functie_baza_items();

/* Creeaza inregistrari implicite cu studenti in baza
   de date la instalarea aplicatiei pe Palm */
    Err functie_baza_students();

/* Creeaza inregistrari implicite cu laboratoare sau
   seminarii in baza de date la instalarea aplicatiei
   pe Palm */
    Err functie_baza_labs();

/* Creeaza inregistrari implicite cu note in baza
   de date la instalarea aplicatiei pe Palm */
    Err functie_baza_note();

/* Creeaza inregistrari implicite cu prezenta studentilor
   in baza de date la instalarea aplicatiei pe Palm */
    Err functie_baza_prez();

/* Creeaza inregistrari implicite cu legaturile dintre
   studenti, discipline etc. in baza de date la
   instalarea aplicatiei pe Palm */
    Err functie_baza_legstuds();

/* Adauga o inregistrare in baza de date */
    Err functie_adauga();

/* Functie care primeste valoarea unui camp */
    Err Get_field_value(UInt16 objectID, char *result);

/* Primeste o inregistrare din baza de date cu adrese */
    Err AddrDBGetRecord(DmOpenRef dbP, UInt16 index,
     DBNotePtr recordP, MemHandle* recordH);
```

```
/* Actualizeaza inregistrarile unui student */
      Err updateStudent(void);

/* sterge o inregistrare de la o anumita adresa */
      Err AddrDeleteRec(DmOpenRef dbP, int data);

/* Modifica o inregistrare in baza de date de adrese */
      Err AddrDBChangeRecord(DmOpenRef dbP,
       UInt16* index, DBNotePtr r);
```

4.3 Instalarea modulelor aplicației *handheld*

Pentru instalarea aplicației *PalmNotes*, trebuie urmați pașii descriși în continuare, cu precizarea că nu este obligatoriu să urmați această secvență în ordinea prezentată.

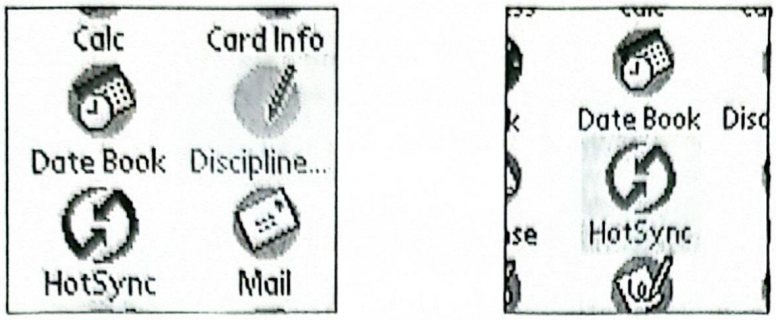
Prezentarea modulelor aplicației

Modulele aplicației rulând pe *handheld* se află în directorul *PalmHandheld*, care conține următoarele subdirectoare:

- *Palm Emulator* – include un emulator pentru testarea aplicației;
- *Palm_Notes_Handheld* – conține aplicația propriu-zisă; această aplicație va fi rulată după ce în prealabil a fost deschis emulatorul și va avea ca efect încărcarea acesteia în memoria dispozitivului *handheld*;
- *Palm_Notes_Conduit* – include două subdirectoare :
  - *ConduitInstall* – director în care se află fișierele necesare instalării conduitei; se va rula *ConduitInstall.exe* pentru a se instala automat conduita în CDK-ul instalat în prealabil.
  - *PalmConduitTransferToFrom* – în acest director se stochează fișierele care controlează modalitatea de transfer a datelor între dispozitivul mobil și calculator; fișierele cu extensia *.dll*, cât și fișierul *config.txt* din *Debug* vor trebui copiate în directorul de unde va rula *HotSync.exe*. Acest director va fi creat la instalarea CDK-ului.
- *PalmNotes* – conține aplicația *desktop* folosită pentru a face managementul situației școlare a studenților. Această aplicație fiind realizată în ASP.NET, presupune să aveți instalat *frameuork*-ul .NET și *Internet Information Services* (sau orice alt server Web). Copiați aplicația în directorul *wwwroot* (considerat implicit rădăcina directoarelor publice expuse de IIS).

Folosirea emulatorului pentru a testa aplicația

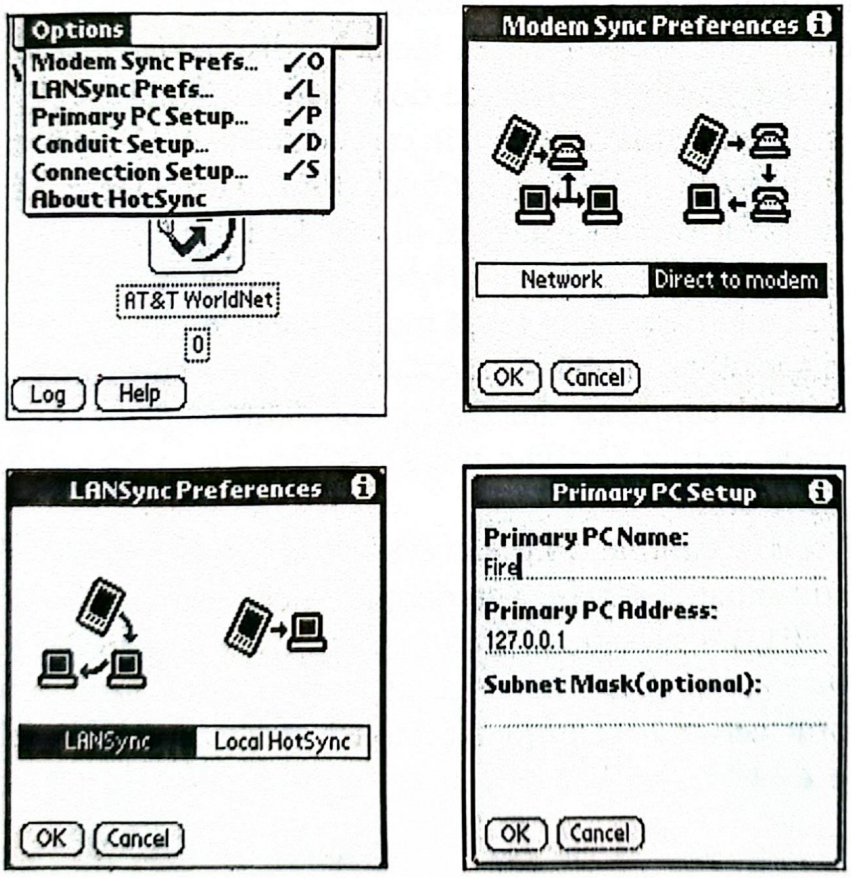
După lansarea aplicației, veți observa pe ecranul emulatorului apariția unei pictograme cu numele „Discipline...", ca în imaginea următoare, acționarea ei determinând lansarea aplicației.



Lansarea aplicației (stânga) și accesarea modulului de sincronizare (dreapta)

După modificarea datelor dorite în cadrul aplicației, trebuie să se realizeze sincronizarea acestora cu calculatorul. Pentru a realiza acest lucru, trebuie să setați corespunzător emulatorul.

Accesul la modulul de sincronizare se face prin intermediul pictogramei *HotSync*. În continuare, veți accesa meniul *Options*. În cadrul acestuia, trebuie să setați pe rând valori pentru *Modem Sync Prefs.*, *LANSync Prefs.* și *Primary PC Setup*.

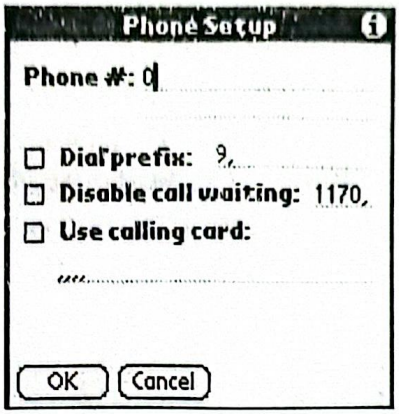


Stabilirea opțiunilor privitoare la sincronizare

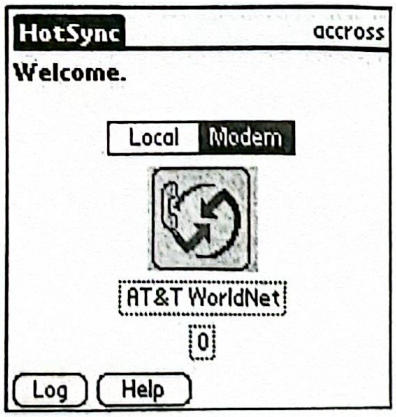
În cadrul meniului *Modem Sync Prefs*... trebuie să selectați opțiunea *Direct to modem*, apoi, din meniul *LAN Sync Prefs*..., trebuie să alegeți opțiunea *LANSync*.

În meniul *Primary PC Setup*... trebuie să introduceți un nume pentru calculatorul *desktop* în câmpul *Primary PC Name*, iar pentru *Primary PC Address* dați 127.0.0.1.

Ultimul pas pe care trebuie să îl mai efectuați este setarea serviciului care va realiza sincronizarea. Acesta se va configura în cadrul meniului *Phone Setup*, trecând 0 pentru numărul de telefon cerut, ca în imaginea alăturată.



În final, testarea se va realiza prin apăsarea pictogramei de sincronizare din meniul principal, ca în figura de mai jos:



Testarea acțiunii de sincronizare

5. Comentarii bibliografice

Pentru primul subcapitol, cititorul interesat de aspectele legate de tehnologia *handheld* Palm OS poate consulta resursele disponibile la adresa *http://www.palmsource.com/palmos/*. Aici se pot găsi prezentări introductive privitoare la tehnologia Palm OS și la principalele elemente care o caracterizează.

O altă resursă importantă pentru persoanele interesate în dezvoltarea de aplicații pentru această platformă o constituie cartea *Palm OS Programming Bible*, scrisă de Lonnon R. Foster și apărută la John Wiley & Sons, 2000. Veți descoperi principalele tehnici de programare pe care trebuie să le adoptați pentru a crea aplicații Palm OS performante. O altă lucrare care poate prezenta interes este N. Rhodes, J. McKeehan, *Palm OS Programming: The Developer's Guide*, O'Reilly & Associates, 1999.

Pentru a înțelege mai bine modalitatea de dezvoltare a unei aplicații Palm OS puteți utiliza mai multe emulatoare, disponibile la adresa *http://www.palmos.com/dev/dl/dl_tools/dl_emulator/*.

În ceea ce privește implementarea sigură a aplicațiilor Web, cititorul poate consulta lucrările A. Kolawa, W. Hicken, C. Dunlop, *Bulletproofing Web Applications*, M&T Books, 2002; B. LaMacchia *et al.*, *.NET Framework Security*, Addison-Wesley, 2002 și J.D.Meier *et al.*, *Building Secure Microsoft ASP.NET Applications: Authentication, Authorization and Secure Communication*, Microsoft Press, 2003.

ANEXA I

ShockWave Flash (.SWF) *versus* Scalable Vector Graphics (.SVG)

Andrei Croitoru, icynoangel@yahoo.com
Cătălin Bulancea, samprasfan03@yahoo.com

Există mai multe tehnologii prin intermediul cărora se poate realiza grafică vectorială bidimensională interactivă pe Web, dintre care s-a evidențiat formatul ShockWave Flash, standardizat de Macromedia, fiind însă un standard al unui singur producător. Din acest motiv, Consorțiul Web (*World Wide Web Consortium*) și-a propus să ofere o modalitate de desenare vectorială bidimensională interactivă pe Web bazată pe XML (*Extensible Markup Language*). Această nouă tehnologie se numește *Scalable Vector Graphics* (*SVG*). Consorțiul Web a introdus SVG în 2001 (recomandarea SVG 1.0), versiunea curentă fiind 1.1 (ianuarie 2003). În prezent se lucrează la standardizarea versiunii SVG 1.2 – detalii la *http://www.w3.org/SVG/*.

Unii specialiști afirmă că SVG este deja la fel de puternic ca Flash sau chiar mai puternic, alții cred că este invers. Ne propunem să realizăm o comparație între aceste două formate (*.swf* și *.svg*.), oferind exemple atât în direcția ShockWave Flash, cât și în direcția Scalable Vector Graphics.

Afirmațiile cu privire la cele două formate se bazează pe specificația formatului de fișier SWF (versiunea 7) și pe recomandarea SVG 1.1. Deoarece comparația de față este doar între *.swf* și *.svg*, există o serie de aspecte suplimentare de care trebuie să ținem seama. În unele cazuri, aceste tehnologii sunt mult prea diferite pentru a fi comparate, deși ele arată și se comportă asemănător. Una dintre deosebiri o constituie faptul că fișierele *.swf* sunt create folosind mediul Flash, pe când pentru formatul *.svg* nu există încă un mediu de dezvoltare complet.

Din motivele enunțate mai sus, vom prezenta doar acele aspecte care pot fi comparate.

Privire de ansamblu

Subiect	SWF	SVG
Standard deschis/ Bazat pe specificații	Nu, licență Macromedia	Da, specificație deschisă
Tipul de format	Binar	Bazat pe text (inteligibil - sintaxă XML)
Tip MIME (*Multipurpose Internet Mail Extension*)	*application/x-shockwave-flash*	*image/svg+xml*
Transferabil ca flux (*streamable*)	Da	Nu
Interactivitate	Da, prin ActionScript și evenimente	Da, prin *scripting*, SMIL și evenimente
Procesare prin DOM (Document Object Model)	DOM intern	SVGDOM în conformitate cu recomandarea DOM nivelul 1
Compresie	Da, *zlib*	Da, *gzip*
Suport nativ pentru navigatoare Web	Nu, dar *plugin*-ul este instalat pe aproape fiecare sistem care rulează sub Windows, Mac OS sau UNIX (Linux)	Nu, dar implementat de câteva navigatoare Web (Mozilla Firefox, Amaya). *Plugin*-ul nu este larg răspândit
Suport pentru stiluri	Specific SWF, subset CSS	Specific SVG, CSS2 și XSL

Suprafața de desenare (Canvas)

Subiect	SWF	SVG
Unități		
Lungime	px	px (implicit), pt, cm, mm, in, pc, %, em
Unghi	grade	grade, radiani
Timp	milisecunde	secunde
Transformări geometrice		
Translatare	Da	Da
Scalare	Da, orizontal și vertical	Da, orizontal și vertical
Rotație	Da	Da
Deformare	Nu, dar posibil prin rotație și scalare	Da, *skewX* și *skewY*
Matrice	Nu	Da
Transformări geometrice imbricate	Da	Da

Figuri geometrice

Subiect	SWF	SVG
Elemente de bază		
Linie	Da, comanda *lineTo*	Da
Linie frântă	Nu, dar posibil de simulat prin multiple comenzi *lineTo*	Da
Dreptunghi	Nu, dar posibil de simulat prin multiple comenzi *lineTo*	Da
Cerc	Nu, dar posibil de simulat prin multiple comenzi *lineTo*/*curveTo*	Da
Elipsă	Nu, dar posibil de simulat prin multiple comenzi *lineTo*/*curveTo*	Da
Poligon	Nu, dar posibil de simulat prin multiple comenzi *lineTo*	Da
Proprietăți avansate		
Segment de arc	Nu, dar posibil de simulat prin multiple comenzi *lineTo*/*curveTo*	Eliptic
Curbe Bézier pătratice	Da, prin *curveTo*	Da
Curbe Bézier cubice	Nu, dar posibil de simulat prin multiple comenzi *lineTo*/*curveTo*	Da
Folosire coordonate absolute/relative	Da	Da

Animații

Subiect	SWF	SVG
Elemente de bază		
Posibilitate de animație	Da, prin script	Da, prin elementele `<animate>`, `<set>`, `<animateMotion>`, `<animateTransform>`, `<animateColor>` și prin *scripting* folosind SVG DOM

Text

Subiect	SWF	SVG
Lucrul cu blocuri de text		
Posibilitate de lucru cu blocuri de text	Da, doar folosind dreptunghiuri, numite „Textfield"	Nu, dar posibil de simulat prin `<tspan>`
Modalităţi de „rupere" a textului (*Textwrapping*)	Da	Nu, dar posibil de simulat prin `<tspan>`
Aliniere	stânga, centru, dreapta	stânga, centru, dreapta
Paragrafe	Da, posibilitate de indentare	Nu
Liste	*Bulleted*	Nu
Bara de defilare	Nu, dar posibil de implementat prin script	Nu, dar posibil de implementat prin script
Fonturi		
Încapsulare fonturi	Da	Da
Dimensiune font	Da	Da
Stiluri	Normal, italic	Normal, italic
Greutate font	Normal, îngroşat	Normal, îngroşat, mai îngroşat, lighter, 100, 200, 300, 400, 500, 600, 700, 800, 900
Deformare font	Nu, dar posibil de simulat prin transformare	normal, wider, narrower, ultra-condensed, extra-condensed, condensed, semi-condensed, semi-expanded, expanded, extra-expanded, ultra-expanded
Stiluri text		
Culoare contur	Da	Da
Culoare fundal	Da	Nu
Decoraţii	Subliniat	Subliniat, linie deasupra, line mediană şi clipire
Spaţiere		
Spaţiere litere	Nu	Da
Spaţiere cuvinte	Nu	Da
Spaţiere linii	Da	Nu
Alte opţiuni		
Selectare text	Da	Da
Tabele	Nu	Nu
Text de-a lungul unei căi	Nu	Da, folosind elementul `<textPath>`

Stiluri şi moduri de afişare

Subiect	SWF	SVG
Tipuri de contururi (Strokes)		
Solid	Da	Da, proprietatea „fill"
Lăţime	Da	Da, proprietatea „stroke-width"
Culoare	Da	Da, proprietatea „fill"
Transparenţă	8-bit alpha	Da, controlabil de la 0 la 1 (implicit 1 - opac)
Line Caps	Rotunde, alte *line caps* se pot simula prin poligoane	*butt, round, square*
Moduri de îmbinare a liniilor (*Line Join*)	Rotunde, alte tipuri se pot simula prin poligoane	De tip *miter, bevel* şi *round*; unghiurile ascuţite şi de tip *miter* se pot controla prin proprietatea „miter-limit"
Linie punctată (*Dashing*)	Nu, dar posibil de simulat prin linii scurte	Da, via „dash-array" şi „dash-offset"
Delimitatori (*Markers*)	Nu, dar posibil de simulat	Da, prin definirea delimitatorilor individuali folosind `<marker>`. Delimitatorii sunt plasaţi la capătul unei linii automat, delimitatorii de start şi de sfârşit pot fi definiţi individual
Şabloane în *Stroke*	Nu	Da, bitmap şi vector
Degradeuri în *Stroke*	Nu	Da
Stiluri diferite într-o singură cale *„multistroking"*	Da, fiecare segment poate conţine un stil diferit	Nu, un element de tip *path* poate avea un singur stil
Modalităţi de umplere (Fills)		
Solid	Da	Da, prin proprietatea „fill-rule" pentru umplerea de poligoane multiple
Degradé (*Gradient*)	Liniar şi radial (până la 8 puncte de control)	Liniar şi radial (nu există limite)
Bitmap	Nu, dar posibil de simulat prin măşti	Da, *bitmap*-uri de tip *single* şi *tiled*
Transparenţă	8-bit alpha	Da, între 0 şi 1 prin proprietatea „opacity"
Şabloane predefinite	Nu, dar posibil de simulat prin obiecte predefinite şi măşti	Da, prin elementul `<pattern>`
Măşti (*Masking*)	Da	Da, orice obiect grafic poate fi folosit ca o mască pentru a îngloba obiectul curent în fundal
Decupare (*Clipping*)	Da	Da, cu `<clipPath>`

Subiect	SWF	SVG
Efecte grafice (filtre)		
Primitive	Flash nu oferă funcţionalitatea aceasta, dar tehnici de colorare şi transformări de transparenţă pot simula unele efecte grafice	Surse de lumină: *distant, point, spot, diffuse, specular* *Blend* *Color Matrix* *ComponentTransfer* *Composite* *ConvolveMatrix* *DisplacementMap* *Flood* *GaussianBlur* *Image* *Merge* *Morphology* *Offset* *Tile* *Turbulence* Remarcă: filtrele SVG pot fi combinate în orice ordine şi ataşate unui obiect

Interactivitate

Subiect	SWF	SVG
Script		
Limbaj de *scripting*	ActionScript 2.0 (conform ECMA-262)	ECMA-262, versiunea 2
Interfaţă de *scripting*	ActionScript API	DOM, nivelul 2
OOP	Da, bazat pe prototipuri	Da, bazat pe prototipuri
Creare dinamică de elemente	Da	Da, prin SVG DOM
Manipulare de elemente/atribute	Da	Da

Subiect	SWF	SVG
Evenimente		
Tip de eveniment	Butoane: *onPress, onRelease, onDragOut, onDragOver, onKeyDown, onKeyUp* *MovieClip*-uri: *onEnterFrame, onLoad, onUnload, onDragOut, onDragOver, onPress, onRelease, onMouseDown, onMouseMove, onMouseUp, onKeyDown, onKeyUp, onSetFocus, onKillFocus* *TextField*-uri: *onChanged, onScroller, onSetFocus, onKillFocus* Altele: *onXML, onConnect, onData, onLoad, onActivity, onSoundComplete*	Interfață: *activate, focusIn, focusOut* Zoom & Pan: *SVGResize, SVGScroll, SVGZoom* Evenimente de stare: *SVGLoad, SVGUnload, SVGAbort, SVGError* Mouse: *click, mouseDown, mouseUp, mouseOver, mouseMove, mouseOut* SMIL/Animații: *beginEvent, endEvent, repeatEvent*
Hiperlegături (*Hyperlinking*)		
Modalități	Da, prin `<a href...>` sau prin „url" și „target" pentru *textlinks*	Da, prin XLink
Legături simple către resurse externe	Da	Da
Legături către elemente interne	Nu, dar posibil de simulat prin ActionScript	Da, se poate face *link* către ID-ul unui element
Legături către un anumit *frame* dintr-o animație	Da	Nu

Exemple

Desenare dreptunghi, redimensionare

- SVG

```
<?xml version="1.0" encoding="utf-8"?>
<!-- creeaza 2 patrate umplute (filled) -->
<svg>
  <rect x="100" y="100" width="30" height="30"/>
  <g transform="scale(2)">
    <rect x="100" y="100" width="30" height="30"/>
  </g>
</svg>
```

- Flash

```
_root.createEmptyMovieClip("rect",0);
with (_root.rect) {
lineStyle(5, 0xff00ff, 100);
moveTo(200, 200);
lineTo(200, 400);
lineTo(400, 400);
lineTo(400, 200);
lineTo(200, 200);
}
_root.rect.duplicateMovieClip("rect1",1);
_root.rect1._xscale = 200;
_root.rect1._yscale = 200;
```

Text

- SVG

```
<text x="10" y="20" style="font:Helvetica;
font-size:15;font-style:bold;stroke-width:5">
exemplu de text</text>
```

- Flash

```
this.textFormat = new TextFormat("Monotype Corsiva", 12,
0xff0000, true, false);
this.createTextField("testText", 10, 300, 300, 100, 20);
testText.text = "Text de test";
testText.selectable = false;
testText.type = "static";
testText.border = true;
testText.setTextFormat(textFormat);
```

Transparență

- SVG

```
<rect x="100" y="100" width="30" height="30" opacity="0.5"/>
```

- Flash

```
_root.myClip._alpha = 50;
```

Gradient

- SVG

```
<defs>
<linearGradient id="Gradient1">
   <stop offset="5%" stop-color="darkblue" />
   <stop offset="100%" stop-color="lightblue" />
</linearGradient>
</defs>
<rect x="100" y="100" width="250" height="30"
      fill="url(#Gradient1)"/>
```

- Flash

```
this.createEmptyMovieClip("gradientMovie", 20);
with (_root.gradientMovie) {
  colors = [0xFF0000, 0x0000FF];
  alphas = [100, 100];
  ratios = [0, 0xFF];
  matrix = {a:200, b:0, c:0, d:0, e:200,
              f:0, g:200, h:200, i:1};
  beginGradientFill("linear",
      colors, alphas, ratios, matrix);
  moveto(100, 100);
  lineto(100, 300);
  lineto(300, 300);
  lineto(300, 100);
  lineto(100, 100);
  endFill();
}
```

Evenimente

- SVG

```
<circle id="round_button" cx="100" cy="100" r="30"
        onclick="ClickButton(evt)"/>
```

- Flash

```
this.buttonName.onRelease = function {
  trace ("Release event !");
}
```

Încărcare imagine

- SVG

```
<image x="20" y="20" width="100" height="130"
       xlink:href="URL"/>
```

- Flash

```
this.createEmptyMovieClip("container", 10);
this.container.loadMovie("URL", 0);
```

Animație

- SVG

```
<rect x="10" y="50" width="30" height="30">
<animate attributeName="x" from="10" to="100"
begin="click" dur="3s" fill="freeze"/>
<animateTransform attributeName="transform"
   attributeType="XML" type="scale" from="1" to="3"
   begin="click" dur="3s" fill="freeze"/>
</rect>
```

- Flash

```
this.rect._x = 10;
this.onEnterFrame = function() {
  if(this.rect._x < 100) {
     this.rect._x += 10;
     this._xscale += 10;
     this._yscale += 10;
  } else {
     delete this.onEnterFrame;
  }
}
```

ANEXA II

PHP *versus* ASP.NET

Bogdan Manolache, toni@MVDmedia.ro
Sabin Buraga, busaco@infoiasi.ro

Ce este PHP?

Atunci când vine vorba de dezvoltarea de aplicații Web, există nenumărate platforme de dezvoltare. PHP este una dintre cele mai populare și utilizate soluții de implementare a aplicațiilor Web.

Serverul de aplicații PHP (*PHP: Hypertext Preprocessor*) inițial a fost gândit ca o simplă aplicație CGI (*Common Gateway Interface*) pentru interpretarea formularelor definite prin HTML și procesate de un program scris într-un limbaj executat pe server. În cazul interfeței CGI, era nevoie de permisiunea de a rula programe pe server, ceea ce conducea uneori la lacune în securitate și, în plus, la disocierea de documentul HTML a programului care procesa datele. PHP este un produs *open-source*. Oricine poate avea acces la codul sursă, putându-l folosi, modifica și apoi redistribui gratis, conform prevederilor stipulate de licența GNU GPL.

PHP se bazează pe preprocesarea paginilor – introducerea de cod interpretabil în cadrul documentelor HTML, cod cuprins între șiruri de caractere specifice. Atunci când preprocesorul rulat pe serverul Web găsește o succesiune de linii de cod PHP, le va interpreta și va realiza acțiunea specificată de program:

```
<?php
    echo "<p>Acest cod a fost interpretat.</p>";
?>
```

Sintaxa PHP este foarte apropiată de cea a limbajelor C și Perl, mai nou împrumutând caracteristici specifice limbajului de programare Java. Mediul PHP oferă suport pentru lucrul cu bazele de date și cu documentele XML, pentru generarea de rapoarte în diverse formate, precum PDF, pentru realizarea de grafică de tip *raster* dinamică etc. O extensie – numită *PHP-GTK* – permite generarea de interfețe cu utilizatorul de sine stătătoare.

Preprocesorul PHP va determina construcţiile PHP delimitate între *tag*-urile „<?php" şi „?>" şi le va interpreta, ignorând restul conţinutului documentului, pe care îl va trimite, la ieşirea standard, nealterat. Astfel, putem privi analizorul PHP ca pe un script CGI care efectuează o preprocesare specială înainte ca datele să fie transmise clientului Web.

PHP oferă un limbaj script puternic şi simplu cu următoarele avantaje: performanţă ridicată, interfeţe de programare pentru diferite sisteme de baze de date, biblioteci încorporate pentru majoritatea sarcinilor Web, disponibilitate gratuită a mediului şi a codului sursă.

De asemenea, există o multitudine de situri Web punând la dispoziţie tutoriale, documentaţii şi colecţii de cod sursă pentru realizarea celor mai frecvente operaţii (de exemplu, implementarea coşului de cumpărături, autentificarea utilizatorilor, lucrul cu diverse tipuri de documente etc.). Anumite clase şi programe importante se regăsesc grupate în PEAR (*PHP Extension and Add-on Repository*).

Ce aduce nou PHP 5?

Programatorii Web care folosesc acum versiunea 4 vor migra – presupunem, în curând – la noua versiune PHP. Versiunea 5, apărută în cursul anului 2004, aduce numeroase îmbunătăţiri, cele mai importante fiind pe partea de programare orientată pe obiect, de lucrul cu bazele de date MySQL şi procesare a fişierelor XML.

Nucleul PHP 5 a fost perfecţionat şi a ajuns la o nouă versiune: *Zend Engine II*. Acesta conţine peste 20 de noi facilităţi care vor înlesni dezvoltarea şi întreţinerea aplicaţiilor Web. Noile îmbunătăţiri şi inovaţii, complet orientate pe obiect, vor contribui la rularea mai rapidă a aplicaţiilor, vor conduce la dezvoltarea obiectuală mult mai intuitivă şi robustă şi vor face din PHP o platformă fezabilă pentru dezvoltarea şi întreţinerea proiectelor mari.

Suportul extins pentru XML impune PHP ca un mediu avansat pentru procesarea documentelor de acest tip. În plus, noua funcţionalitate *SimpleXML* permite dezvoltatorilor să manipuleze documentele XML ca şi cum ar fi obiecte PHP, într-un mod similar celui din Perl. Aceasta înseamnă că programatorii vor putea folosi documentele XML fără a avea nevoie de cunoştinţe avansate în domeniu.

Pentru a asigura programatorilor posibilitatea integrării de servicii Web în aplicaţiile folosite, PHP 5 include un modul SOAP (*Simple Object Access Protocol*), care permite interoperabilitatea cu serviciile Web (eventual, dezvoltate în alte limbaje sau pe alte platforme), utilizându-se o interfaţă de programare uşor de exploatat.

Mai mult decât atât, și interfața de programare cu bazele de date – în special cu serverul MySQL – a fost îmbunătățită, pachetul PHP incluzând un server de baze de date minimal, util pentru testarea codului și/sau exploatarea unor situri Web de mici dimensiuni, fără a mai trebui instalate aplicații suplimentare.

În concluzie, PHP 5 asigură viitorul acestei platforme – rulând pe sisteme de operare multiple – cu numeroasele îmbunătățiri aduse versiunii precedente.

Ce este ASP.NET?

Abordare similară PHP-ului, ASP (*Active Server Pages*) este produs de Microsoft începând cu anul 1996, cu scopul de a oferi dezvoltatorilor Web o alternativă orientată pe obiect, aplicațiile putându-se construi ușor în limbaje de *scripting* ca JScript sau VBScript. ASP este însă destul de rudimentar și tributar serverului IIS sub Windows, cu toate că există instrumente de migrare a codului și pe alte platforme. Ca și la PHP, codul sursă poate fi introdus între delimitatori speciali (aici, se folosesc „<%" și „%>"), mediul ASP oferind o serie de obiecte predefinite pentru accesul la variabilele serverului Web, managementul sesiunilor și al *cookie*-urilor, generarea de conținut trimis navigatorului etc. Serverele ASP oferă o conectivitate strânsă cu diverse servere de baze de date și permit execuția de componente ActiveX.

Succesorul său – ASP.NET – este o parte componentă a familiei de produse/tehnologii Microsoft .NET și reprezintă un instrument puternic pentru realizarea de aplicații Web dinamice. ASP.NET lucrează cu limbaje de tip *script* (interpretate), ca VBScript, JScript, Perl sau Python, precum și cu limbaje compilate, cum ar fi C#, C++ ori Visual Basic. ASP.NET este complet orientat pe obiect, oferind facilități avansate în această privință.

La fel ca în cazul unei aplicații normale, codul ASP.NET este procesat pe server. După ce codul a fost procesat, serverul returnează clientului codul HTML rezultat. În cazul în care clientul are suport pentru JavaScript, atunci serverul va folosi această facilitate pentru a permite utilizatorului o navigare mai rapidă și facilă. Chiar dacă HTML este factorul limitator, ASP.NET permite scrierea de cod obiectual pur pentru aplicațiile Web. În funcție de navigator, mediul ASP.NET va genera cod sursă HTML (plus, eventual, JavaScript) specific acestuia, accelerând scrierea de aplicații Web.

ASP.NET permite implementarea și exploatarea serviciilor Web, bazându-se pe ierarhia de clase pusă la dispoziție de platforma .NET.

Ca și alte medii, ASP.NET facilitează lucrul cu fișierele XML pentru stocarea datelor, manipularea datelor sau diverse configurări. Instrumentele care sunt incluse în ASP.NET pentru lucrul cu XML sunt foarte ușor de utilizat, pentru

aceasta folosindu-se o interfață de programare simplificată, diferită de maniera de procesare prin DOM ori SAX (*Simple API for XML*).

Un mare avantaj pe care îl oferă ASP.NET este biblioteca de clase predefinite, încorporând un număr impresionant de funcționalități generale sau specializate (ca, de exemplu, internaționalizarea aplicațiilor, procesarea tranzacțiilor, lucrul cu fire de execuție sincronizate sau manipularea avansată a colecțiilor de date).

Microsoft a realizat o documentație foarte cuprinzătoare pentru ASP.NET, precum și pentru toate limbajele .NET: informații despre clase, inclusiv exemple pentru fiecare clasă, funcție, metodă sau proprietate accesibilă din ASP.NET. Ca și la PHP, și în acest caz există disponibile pe Web diverse situri oferind – gratuit sau contra cost – colecții de componente care pot fi utilizate în cadrul siturilor Web concepute în ASP.NET.

Drept concluzie, ASP.NET poate fi considerat un mediu de dezvoltare a aplicațiilor Web complet obiectuale pentru realizarea de proiecte complexe, beneficiind de suportul acordat de platforma .NET, în special pentru Windows.

Ce alegem?

Considerând că încă nu v-ați decis ce alegere să faceți, vom prezenta, în continuare, un tabel comparativ între PHP (versiunile 4 și 5) și ASP.NET.

	PHP 4	PHP 5	ASP.NET
Disponibilitatea aplicației	gratuită	gratuită	gratuită
Disponibilitatea platformei	gratuită	gratuită (contra cost pentru *ZendStudio*)	contra cost (gratuită pentru *WebMatrix*)
Viteză	mare	mare	(relativ) mare, în funcție de limbajul folosit
Eficiență	mare	mare	mică
Securitate	mare	mare	mare
Stabilitate	da	da	da
Disponibilitatea surselor	da	da	nu
Suport pentru excepții	nu	da	da
Suport pentru baze de date	da	da (inclusiv noua interfață `mysqli`)	da (via ADO.NET)
Suport pentru procesarea documentelor XML	da (DOM, SAX)	da (DOM, SAX, simplificat)	da (simplificat)

	PHP 4	PHP 5	ASP.NET
Suport pentru programarea obiectuală (OOP)	(relativ) slab	puternic	excelent
Suport pentru controale I/O pe partea de server	nu	nu	da (*WebForms*)
Suport nativ pentru generarea altor tipuri de conținuturi (PDF, SWF,...)	da	da	nu
Componente predefinite (de exemplu, calendar)	nu, dar disponibile via PEAR	nu, dar disponibile via PEAR	da, plus disponibile pe Web (gratuite ori în variantă comercială)
Delimitare clară între logica aplicației, date și prezentare	nu	parțială	da
Suport pentru mai multe limbaje de programare (multi-limbaj)	nu	nu	da
Rulează pe servere Web multiple (multi-platformă)	da (Apache, IIS etc.)	da (Apache, IIS etc.)	nu (doar IIS) da (proiectul *open-source* Mono)
Suport pentru dezvoltarea de proiecte de anvergură	(relativ) slab	mediu (spre puternic)	puternic

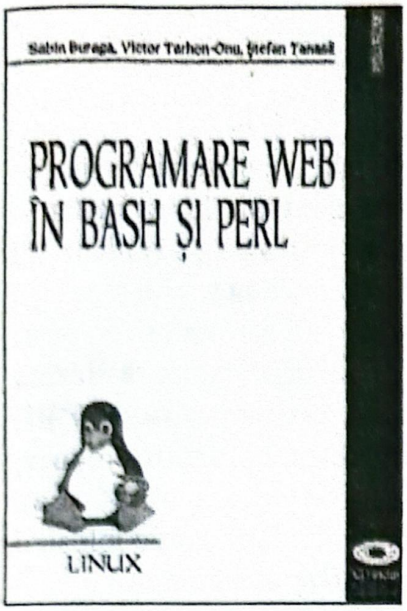
Bibliografie generală

1. D. Amor, *The E-Business (R)evolution*, Prentice Hall, 1999
2. T. Archer, *Inside C#*, Microsoft Press, 2002
3. J. Bowman, S. Emerson, M. Darnovsky, *The Practical SQL Handbook: Using Structured Query Language*, Addison-Wesley, 1996
4. J. Bradenbaugh, *JavaScript Application Cookbook*, O'Reilly & Associates, 1999
5. B. Breedlove, *Web Programming Unleashed*, Sams, 2003
6. E. Brown, *Windows Forms Programming with C#*, Manning, 2002
7. M. Brut, S. Buraga, *Prezentări multimedia pe Web*, Polirom, Iaşi, 2004: *http://www.infoiasi.ro/~multimedia/*
8. B. Bulger, J. Greenspan, *MySQL/PHP Database Applications*, Wiley, 2004
9. S. Buraga, *Proiectarea siturilor Web*, Polirom, Iaşi, 2002: *http://www.infoiasi.ro/~design/*
10. S. Buraga, *Semantic Web. Fundamente şi aplicaţii*, Matrix Rom, Bucureşti, 2004: *http://www.infoiasi.ro/~sweb/*
11. S. Buraga, *Tehnologii Web*, Matrix Rom, Bucureşti, 2001: *http://www.infoiasi.ro/~busaco/books/web.html*
12. S. Buraga, V. Tarhon-Onu, Ş. Tanasă, *Programare Web în bash şi Perl*, Polirom, Iaşi, 2002: *http://www.infoiasi.ro/~cgi/*
13. S. Buraga (coord.), *Aplicaţii Web la cheie. Studii de caz implementate în PHP*, Polirom, Iaşi, 2003: *http://www.infoiasi.ro/~phpapps/*
14. A. Chaudhri, A. Rashid, R. Zicari, *XML Data Management: Native XML and XML-Enabled Database Systems*, Addison-Wesley, 2003
15. R. Chun, *Macromedia Flash MX Advanced for Windows and Macintosh: Visual QuickPro Guide*, Peachpit Press, 2002
16. C. Cosentino, *Advanced PHP for Web Professionals*, Prentice Hall, 2003
17. J. Davies, D. Fensel, F. van Harmelen (eds.), *Towards the Semantic Web*, John Wiley & Sons, 2003
18. B. Daum, U. Merten, *System Architecture with XML*, Elsevier Science, 2003
19. C. Doctorow *et al.*, *Essential Blogging*, O'Reilly & Associates, 2002
20. K. Egervari, J. Parise, J. Coggershall, *Professional PHP 4 Programming*, Wrox Press, 2002
21. D. Esposito, *Building Web Solutions with ASP.NET and ADO.NET*, Microsoft Press, 2002
22. D. Flanagan, *JavaScript: The Definitive Guide*, O'Reilly & Associates, 1998
23. M. Fotache, *SQL*, Polirom, Iaşi, 2001
24. R. Fournier, *A Methodology for Client/Server and Web Application Development*, Prentice Hall, 1998
25. W. Galitz, *The Essential Guide to User Interface Design*, John Wiley & Sons, 1997
26. I. Gilfillan, *Mastering MySQL 4*, Sybex, 2003
27. D. Goodman, *JavaScript Bible*, IDG Books, 1998
28. C. Gorman, *Programming Web Services with SOAP*, O'Reilly & Associates, 2001
29. B. Hartman *et al.*, *Mastering Web Services Security*, Wiley Publishing, 2003
30. M. Kabir, *Apache Server 2 Bible*, Hungry Minds, 2002
31. R. Katz, *Web Portals and Higher Education*, Educause, 2003
32. A. Kolawa, W. Hicken, C. Dunlop, *Bulletproofing Web Applications*, M&T Books, 2002

33. B. LaMacchia *et al.*, *.NET Framework Security*, Addison-Wesley, 2002
34. K. Laudon, C. Traver, *E-Commerce: Business, Technology, Society*, Addison-Wesley, 2003
35. R. Lerdorf, K. Tatroe, Bob Kaehms, R. McGredy, *Programming PHP*, O'Reilly & Associates, 2002
36. D. Lucanu, *Bazele proiectării programelor și algoritmilor*, Editura Universității „Al.I. Cuza", Iași, 1996
37. F. Manola, E. Miller (eds.), *RDF Primer*, W3C Recommendation, Boston, 2004: *http://www.w3.org/TR/rdf-primer/*
38. J.D. Meier *et al.*, *Building Secure Microsoft ASP.NET Applications: Authentication, Authorization, and Secure Communication*, Microsoft Press, 2003
39. J. Meloni, *PHP Fast & Easy Web Development*, Premier Press, 2002
40. C. Pribeanu (coord.), *Introducere în interacțiunea om-calculator*, Matrix Rom, București, 2003
41. J. Prosise, *Programming Microsoft .NET*, Microsoft Press, 2002
42. N. Rhodes, J. McKeehan, *Palm OS Programming: The Developer's Guide*, O'Reilly & Associates, 1999
43. W. Richardson, D. Avondolio, *Professional Portal Development*, Wiley, 2004
44. M. Sârbu, Articole publicate pe Web: *http://www.intraweb.ro/txt/Articole*
45. N. Shi, V.K. Murthy, *Architectural Issues of Web-Enabled Electronic Business*, Idea Group Publishing, 2003
46. C. Shiflett, *HTTP Developer's Handbook*, Sams Publishing, 2003
47. A. Skonnard, M. Gudgin, *Essential XML Quick Reference*, Addison-Wesley, 2002
48. S. Suehring, *MySQL Bible*, Wiley Publishing, 2002
49. P. Thorsteinson, G. Ganesh, *.NET Security and Cryptography*, Prentice Hall PTR, 2003
50. A. Trachtenberg, *Upgrading to PHP 5*, O'Reilly & Associates, 2004
51. Ș. Trăușan-Matu *et al.*, *Prelucrarea documentelor folosind XML și Perl*, Matrix Rom, București, 2001
52. A. Turtschi *et al.*, *C# .NET Web Developer's Guide*, Syngress Publishing, 2002
53. C. Vârlan, *Macromedia Flash*, Polirom, Iași, 2004: *http://www.infoiasi.ro/~flash/*
54. L. Wall *et al.*, *Programming Perl*, O'Reilly & Associates, 2000
55. L. Welling, *PHP and MySQL Web Development*, Sams, 2003
56. H.E. Williams, D. Lane, *Web Database Applications with PHP & MySQL*, O'Reilly & Associates, 2002
57. K. Williams, *Professional XML Databases*, Wrox Press, 2000
58. M. Williams, *Microsoft Visual C# .NET*, Microsoft Press, 2002
59. ***, *123Aspx*: *http://www.123aspx.com/*
60. ***, *ActionScript*: *http://www.actionscript.org/*
61. ***, *ActionScript Toolbox*: *http://actionscript-toolbox.com/*
62. ***, *Apache2Triad*: *http://apache2triad.sourceforge.net/*
63. ***, *Apache Week*: *http://www.apacheweek.com/*
64. ***, *ASP.NET Resource Index*: *http://www.411asp.net/*
65. ***, *ASP.NET WebMatrix*: *http://www.asp.net/WebMatrix*
66. ***, *ASP-PHP*: *http://www.asp-php.net/*
67. ***, *Bad User Interface Design*: *http://www.baddesigns.com/*
68. ***, *Blogger*, *http://www.blogger.com/*
69. ***, *Browser Watch*, *http://www.browserwatch.com/*
70. ***, *CiteSeer*: *http://citeseer.nj.nec.com/cs*
71. ***, *CodeGuru*: *http://www.codeguru.com/*
72. ***, *Code Project*: *http://www.codeproject.com/*

73. * * *, *CoverPages*: *http://www.coverpages.org/*
74. * * *, *CPAN (Comprehensive Perl Archive Network)*: *http://www.cpan.org/*
75. * * *, *Data Model*: *http://www.datamodel.org/*
76. * * *, *Database Forums*: *http://www.dbforums.com/*
77. * * *, *Database Journal*: *http://www.databasejournal.com/*
78. * * *, *Dave Beckett's Resource Description Framework (RDF) Resource Guide*: *http://www.ilrt.bris.ac.uk/discovery/rdf/resources/*
79. * * *, *DataBase Magazine*: *http://www.dbazine.com/*
80. * * *, *Developmentor*: *http://www.develop.com/tutorials/*
81. * * *, *Dev Shed*: *http://www.devshed.com/*
82. * * *, *Dublin Core Metadata Initiative*: *http://www.dublincore.org/*
83. * * *, *ECMAScript Specification*: *http://www.ecma.ch/*
84. * * *, *Electronic Commerce on Internet*: *http://e-comm.internet.com/*
85. * * *, *Flash Kit*: *http://www.flashkit.com/*
86. * * *, *Flash MX*: *http://www.flash-mx.com/*
87. * * *, *Free On-line Dictionary of Computing*: *http://www.foldoc.org/*
88. * * *, *Free Perl Code*: *http://www.freeperlcode.com/*
89. * * *, *Fixing Your Web Site*: *http://www.fixingyourwebsite.com/*
90. * * *, *Got .NET*: *http://www.gotdotnet.com/*
91. * * *, *HCI Bibliography*: *http://www.hcibib.org/*
92. * * *, *IBM Developer's Site*: *http://www.ibm.com/developer/*
93. * * *, *JavaScript Examples*: *http://www.js-examples.com/*
94. * * *, *JavaScript Resources*: *http://dmoz.org/Computers/Programming/Languages/JavaScript/*
95. * * *, *Journal of Web Semantics*: *http://www.semanticwebjournal.org/*
96. * * *, *Learning to Use Regular Expressions*: *http://gnosis.cx/publish/programming/regular_expressions.html*
97. * * *, *Macromedia Developers Network*: *http://www.macromedia.com/devnet/*
98. * * *, *Microsoft Developers Network*: *http://msdn.microsoft.com/*
99. * * *, *Microsoft Scripting Technologies*: *http://msdn.microsoft.com/scripting/*
100. * * *, *Mobile/Wireless Development for Palm OS Technologies*: *http://www.developer.com/ws/palm/*
101. * * *, *Mono Project*: *http://www.mono-project.com/*
102. * * *, *MySQL Central*: *http://www.mysql.com/*
103. * * *, *Netscape's JavaScript Guide*: *http://devedge.netscape.com/library/manuals/2000/javascript/1.3/guide*
104. * * *, *Nic's JavaScript Page*: *http://www.javascript-page.com/*
105. * * *, *Oasis*: *http://www.oasis-open.org/*
106. * * *, *O'Reilly OnLAMP*: *http://www.onlamp.com/*
107. * * *, *O'Reilly OnDotNET*: *http://www.ondotnet.com/*
108. * * *, *Palm OS*: *http://www.palmos.com/*
109. * * *, *Palm Source*: *http://www.palmsource.com/palmos/*
110. * * *, *Palm Universe*: *http://www.palm.com/*
111. * * *, *PalmPower On-line Magazine*: *http://www.palmpower.com/*
112. * * *, *PEAR (PHP Extension and Application Repository)*: *http://pear.php.net/*
113. * * *, *Perl*: *http://www.perl.com/*
114. * * *, *Perl Monks*: *http://www.perlmonks.com/*
115. * * *, *Perl XML*: *http://www.perlxml.com/*
116. * * *, *PHP*: *http://www.php.net/*

117. * * *, *PHP Builder*: *http://phpbuilder.com/*
118. * * *, *PHP Classes*: *http://www.classes.net/*
119. * * *, *PHP Developer*: *http://www.phpdeveloper.org/*
120. * * *, *PHP World*: *http://www.phpworld.com/*
121. * * *, *RDFweb*: *http://rdfweb.org/*
122. * * *, *RichInStyle*: *http://www.richinstyle.com/*
123. * * *, *Road Coders*: *http://www.roadcoders.com/*
124. * * *, *Romanian Web Developers*: *http://rowd.zuavra.net/*
125. * * *, *SAX (Simple API for XML) Project*: *http://www.saxproject.org/*
126. * * *, *Scripts*: *http://www.worldwidemart.com/scripts/*
127. * * *, *SitePoint*: *http://www.sitepoint.com/*
128. * * *, *SOAP Software*: *http://www.soapware.org/*
129. * * *, *Source Forge*: *http://www.sourceforge.net/*
130. * * *, *Special Interest Group on Semantic Web and Information Systems – SIGSEMIS*: *http://www.sigsemis.org/*
131. * * *, *StudentClub.ro*: *http://www.studentclub.ro/*
132. * * *, *Sun XML Developer Connection*: *http://www.sun.com/software/xml/developers/*
133. * * *, *SVG Demos*: *http://adobe.com/svg/demos/main.html*
134. * * *, *SVG Zone, Adobe*: *http://www.adobe.com/svg/*
135. * * *, *SWF.ro*: *http://www.swf.ro/*
136. * * *, *The JavaScript Source*: *http://javascript.internet.com/*
137. * * *, *The Perl Journal*: *http://www.tpj.com/*
138. * * *, *The Site Wizard*: *http://www.thesitewizard.com/*
139. * * *, *The XML Cover Pages*: *http://xml.coverpages.org/*
140. * * *, *Usable Web*: *http://usableweb.com/*
141. * * *, *useit.com Jakob Nielsen's site*: *http://www.useit.com/*
142. * * *, *WAP Forum*: *http://www.wapforum.org/*
143. * * *, *WAP Links*: *http://www.waplinks.com/*
144. * * *, *Web Compare*: *http://www.webcompare.com/*
145. * * *, *Web Developer*, *http://www.webdeveloper.com/*
146. * * *, *Webmaster World*: *http://www.webmasterworld.com/*
147. * * *, *Web Reference*, *http://www.webreference.com/*
148. * * *, *Web Services*, *http://www.webservices.org/*
149. * * *, *Web Techniques*: *http://www.webtechniques.com/*
150. * * *, *WebmasterBase*: *http://www.webmasterbase.com/*
151. * * *, *World Wide Web Consortium*: *http://www.w3.org/*
152. * * *, *Wireless Developers Network*: *http://www.wirelessdevnet.com/*
153. * * *, *XMethods*: *http://www.xmethods.com/*
154. * * *, *XML.com*: *http://xml.com/*
155. * * *, *XML.org*: *http://www.xml.org/*
156. * * *, *XML Software*: *http://www.xmlsoftware.com/*
157. * * *, *XML Standards Library*: *http://www.mobileworkspace.com/xmlstds/*
158. * * *, *XML Tree*: *http://www.xmltree.com/*
159. * * *, *Zend – the PHP Company*: *http://www.zend.com/*
160. * * *, *ZVON*: *http://www.zvon.org/*

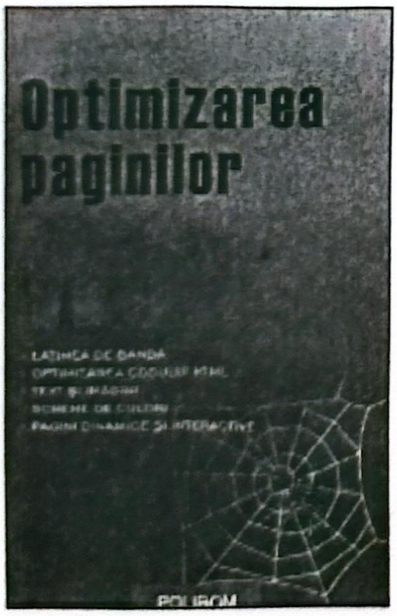


Sabin Buraga, Victor Tarhon-Onu, Ştefan Tanasă

PROGRAMARE WEB ÎN BASH ŞI PERL

Lucrarea descrie în detaliu unul dintre cele mai folosite standarde de programare pe partea de server a aplicaţiilor destinate Web-ului – CGI (*Common Gateway Interface*), punând un accent deosebit asupra elaborării de aplicaţii CGI în două dintre limbajele de tip *script*, larg utilizate pe platforma Linux: *shell*-ul bash şi limbajul Perl. Autorii insistă asupra prelucrării practice a datelor prin intermediul serverelor de baze de date relaţionale (de exemplu, MySQL sau PostgreSQL) şi a documentelor XML (*Extensible Markup Language*).
CD-ul care însoţeşte lucrarea cuprinde atât codurile-sursă ale majorităţii exemplelor şi studiilor de caz prezentate, cât şi documentaţii suplimentare, programe şi aplicaţii utile pentru experimentarea şi aprofundarea materialului din carte, disponibile pentru diferite distribuţii Linux sau alte platforme.

Protocolul HTTP (*HyperText Transfer Protocol*) • Scheme XML • Limbajele de marcare şi aplicaţiile lor • Standardul CGI (*Common Gateway Interface*) • *Cookie*-urile • Programarea CGI în bash • Scripturi CGI în bash • Limbajul şi modulele Perl • Programarea CGI în Perl • Perl şi bazele de date relaţionale • Prelucrarea documentelor XML • Migrarea de la baze de date relaţionale la documente XML

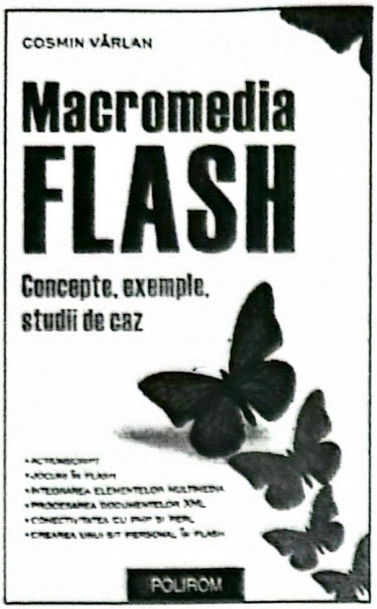


Călin Ioan Acu

OPTIMIZAREA PAGINILOR WEB

O pagină Web trebuie să stârnească interesul. Conţinutul ei nu se poate reduce la câteva fraze scurte, dar nici nu poate fi un amalgam de sunete şi imagini. Situl trebuie să comunice mult, cu mijloace puţine. Trebuie să fie afişat imediat, dar să fie spectaculos şi atractiv. Cum să navigăm rapid şi fără sincope? Ce modificări trebuie aduse sitului pentru a-l optimiza şi a-l face mai uşor de accesat? Iată doar o parte din întrebările la care volumul încearcă să ofere un răspuns, structurat astfel încât să fie accesibil şi începătorilor (pentru care există explicaţii detaliate, exemple practice şi un cuprinzător studiu de caz), şi celor avansaţi, care doresc să-şi verifice şi valideze soluţiile folosite.

Lăţimea de bandă • Optimizarea codului HTML • Utilizarea şi reutilizarea imaginilor • Text sau imagini? • Divizarea imaginilor • Utilizarea *thumbnails*-urilor



Cosmin Vârlan

MACROMEDIA FLASH

Cartea descrie una dintre cele mai populare şi de succes tehnologii pentru crearea de conţinut hipermedia – Macromedia Flash. Conceput într-un stil agreabil şi accesibil, materialul abundă în exemple sugestive şi soluţii practice, indispensabile unor actuali sau viitori creatori de situri Web. Volumul oferă detalii privitoare la realizarea unor *gadget*-uri Flash clasice, precum *preloader*-ul, efectul de *fading* sau animarea meniurilor, jocuri, culminând cu un studiu de caz ce are drept scop construirea unui sit personal, configurabil graţie folosirii limbajului XML (*Extensible Markup Language*).

Realizarea elementelor grafice • Lucrul cu simboluri • Realizare de efecte grafice • Dezvoltarea programelor în ActionScript • Controlul liniei temporale • Suportul pentru navigare, reţea, tipărire • Manipularea *MovieClip*-urilor • Construirea claselor • Utilizarea şi realizarea componentelor • Crearea de instrumente grafice • Interfeţe Flash pentru programe Visual Basic • Realizarea unui joc, a unui album foto, a unei aplicaţii pentru grafică • Realizarea unui sit Web

www.polirom.ro

Redactor: Mădălina Iordache
Coperta: Ionuţ Broştianu

Bun de tipar: octombrie 2004. Apărut: 2004
Editura Polirom, B-dul Carol I nr. 4 • P.O. Box 266
700506, Iaşi, Tel. & Fax: (0232) 21.41.00 ; (0232) 21.41.11 ;
(0232) 21.74.40 (difuzare) ; E-mail : office@polirom.ro
Bucureşti, B-dul I.C. Brătianu nr. 6, et. 7, ap. 33,
O.P. 37 • P.O. Box 1-728, 030174
Tel. : (021) 313.89.78 ; E-mail : polirom@dnt.ro

Tiparul executat la S.C. LUMINA TIPO s.r.l.
str. Luigi Galvani nr. 20 bis, sect. 2, Bucureşti
Tel./Fax : 211.32.60, 212.29.27, E-mail : lumina-lex@fx.ro

VOLUM COORDONAT DE
SABIN BURAGA

Situri WEB la cheie

SOLUTII PROFESIONALE DE IMPLEMENTARE

Cartea realizează o radiografie a tehnologiilor actuale de programare Web, familiarizând cititorul cu metodologiile şi cele mai bune practici în dezvoltarea unor aplicaţii concrete, de anvergură. Materialul continuă tematica abordată de volumul *Aplicaţii Web la cheie. Studii de caz implementate în PHP*, descriind o paletă mai largă de limbaje şi paradigme de programare – PHP, ActionScript, C#, Perl, JavaScript etc. Studiile de caz prezentate detaliază etapele de proiectare, implementare şi exploatare ale unor tipuri reprezentative de situri Web, cele nouă aplicaţii ilustrând maniera de utilizare a tehnologiilor Web de ultimă oră, într-o varietate de domenii (multimedia, *e-business*, comunităţi virtuale, *e-learning*, Web semantic, *wireless* etc.). Lucrarea reprezintă un instrument de lucru indispensabil actualilor şi viitorilor specialişti IT, în general, şi dezvoltatorilor Web, în special.

Procesarea documentelor XML • Realizarea unei enciclopedii în Flash • Crearea de interfeţe Web prin SVG • Dezvoltarea portalurilor în PHP • Managementul *blog*-urilor în Perl • Fotbal virtual prin ASP.NET • De la *desktop* la dispozitive *handheld* • Utilizarea metadatelor • Aplicaţii *e-learning* • Interacţiunea dintre Flash şi PHP • Modelarea relaţiilor dintre resurse • Inginerie Web • SWF *vs.* SVG • Folosirea procedurilor stocate în contextul serviciilor Web • Arhitectura unei aplicaţii Palm OS • PHP *vs.* ASP.NET

Editura POLIROM
www.polirom.ro

ISBN 973-681-785-7
9 789736 817854